Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

Arquitectura de Alta Disponibilidad para Centrales Telefónicas de VoIP basado en Software Libre

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por los Bachilleres:

Alejandro C. Martin R.
C.I.: 21132415
alejandroc.martinr@gmail.com

Jesús I. Gómez P.
C.I.: 19335379
jesus.igp009@gmail.com

Para optar al título de Licenciado en Computación

Tutores**:** Prof. Dedaniel Urribarri y Prof. Eric Gamess

Caracas, Octubre 2016

Universidad Central de Venezuela

Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

ACTA DE VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los Bachilleres Alejandro Martín C.I. V-21.132.415 y Jesús Gómez C.I. V-19.335.379, con el título "**Arquitectura de Alta Disponibilidad para Centrales Telefónicas VoIP en Software Libre**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el nombrado trabajo por cada uno de los miembros del jurado, éste fijó el día martes 25 de octubre de 2016, para que sus autores lo defendieran en forma pública, en la Sala I de la Escuela de Computación, mediante una exposición oral de su contenido, luego de la cual respondieron satisfactoriamente a las preguntas que les fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela.

Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

Es de aclarar que el Profesor Eric Gamess se encuentra de permiso fuera del país. Por esta razón, estuvo via videoconferencia, de común acuerdo con los demás miembros del jurado. Por ende, el Profesor Robinson Rivas, director de la Escuela de Computación, firma el presente documento en su lugar.

En fe de lo cual se levanta el presente acta en Caracas a los 25 días del mes de octubre del año 2016, dejando constancia que el Profesor Dedaniel Urribarri actuó como coordinador del jurado.

Prof. Dedaniel Urribarri
Tutor

por: Robinson Rivas

Prof. Eric Gamess
Tutor

REPUBLICA DE VENEZUELA
Facultad de Ciencias
Escuela de Computación
Universidad Central - Caracas

Prof. Roger Bello
Jurado Principal

Prof. Antonio Russoniello
Jurado Principal

Dedicatoria

Este Trabajo Especial de Grado va dedicado a nuestros padres, madres y hermanos por su ayuda y amor incondicional, por su apoyo a lo largo de toda nuestra trayectoria universitaria, sin ustedes no hubiera sido posible nuestro crecimiento personal y profesional, sin ustedes no hubiéramos podido cumplir con nuestros objetivos y metas.

Agradecimientos

Alejandro Martin

Quiero agradecer principalmente a aquellas personas que hayan ayudado a lo largo de la realización de este Trabajo y de mi carrera de licenciatura.

Agradezco a mi madre Marbella por el amor y el apoyo para sacar siempre lo mejor de mí y ayudar a desarrollarme como persona.

A mi padre Raúl por su paciencia, su criterio y buenos consejos que me ayudaron para lograr siempre mis objetivos.

A mi hermano Mariano por su eterna compañía y solidaridad.

A mi tutor Dedaniel Urribarri por aconsejarme a lo largo del Trabajo para lograr los objetivos y brindarme el apoyo de su empresa para completar dicho Trabajo.

A mi tutor Eric Gamess por su experiencia y apoyo para completar un Trabajo de calidad.

A mi amiga Gabriela por su amistad y apoyo incondicional a lo largo de la carrera y más allá.

A mi compañero de trabajo Jesús para lograr cumplir los objetivos de este Trabajo.

A la mejor universidad del mundo, la Universidad Central de Venezuela por permitir formarme como Profesional.

Sin tener una mención principal, quiero agradecerles a todas aquellas personas que a lo largo de mi carrera hicieron que ésta sea de las mejores experiencias vividas

Jesús Gómez

Agradezco a Dios antes que a nada por permitirme estar donde debo estar, por proveerme de temple y perseverancia para cumplir con mis metas y objetivos, por mostrarme un camino lleno de sorpresas y felicidades.

Agradezco a mi madre y a mi padre por su apoyo y ayuda incondicional, por siempre estar presentes y hacerme fuerte cuando el tiempo y las circunstancias así lo requirieron.

A mi esposa e hija por ser la luz que marca mi futuro, por ser mi apoyo y felicidad en la vida, a toda los comunidad universitaria y científica, a nuestros tutores Dedaniel Urribarri y Eric Gamess por siempre mostrarnos lo bueno y lo malo, por ayudarnos a mejorar

A mi compañero de trabajo Alejandro Martin por siempre estar presente y no renunciar a esta meta.

A todos Gracias.

Resumen

TÍTULO
Arquitectura de Alta Disponibilidad para Centrales Telefónicas de VoIP basado en Software Libre

AUTORES
Alejandro C. Martin R.
Jesús I. Gómez P.

TUTORES
Prof. Dedaniel Urribarri y Prof. Eric Gamess.

Resumen

El avance de la tecnología en el campo de comunicaciones impulsa al crecimiento de algunos conceptos nuevos y existentes, llevándolos a formar parte de infraestructuras de red de computadoras con la capacidad de ofrecer variedades de servicios. De la misma manera, las organizaciones, gracias a este crecimiento, apuestan a la implementación o consumo de dichos servicios con el objetivo de aumentar la eficiencia de operación.

Una de estas tecnologías es la de VoIP, esta se encuentra disponible en todo el mundo para la transmisión de voz y video sobre redes de computadores, con el objetivo de proveer un servicio más eficiente y flexible que las ofrecidas por tecnologías de telefonía convencional. Como cualquier servicio que se puede proveer en el campo de redes de computadores, las arquitecturas que se encargan de proveerlas pueden sufrir fallas que limitan el acceso a dicho servicio y contienen limitaciones de escalabilidad basadas en sus diseños, es por ello que en este Trabajo Especial de Grado se propone la implementación de una Arquitectura de Alta Disponibilidad para Centrales Telefónicas de VoIP basado en Software Libre.

Dicha arquitectura consta de herramientas como Asterisk, para el establecimiento de comunicaciones principalmente en el área de VoIP y debido a sus limitaciones de escalabilidad y alta disponibilidad, se combina con herramientas como Kamailio SIP Server el cual ofrece mecanismos de contingencia y balanceo de carga, para la implementación de una arquitectura confiable en caso de fallas. Además, se aprovecha la utilización de herramientas como PaceMaker/Corosync para la creación de infraestructuras de clustering y gestión de servicios para el monitoreo y el ofrecimiento de mecanismos de contingencia a nivel de servicios y nodos dentro de la arquitectura.

En este trabajo se realiza la implementación de una arquitectura de telefonía que, además de ofrecer los mecanismos anteriormente mencionados, ofrece mecanismos para la comunicación de usuarios en otras arquitecturas de red diferentes. Una vez implementada la arquitectura, se propondrán distintos escenarios de prueba para la verificación del comportamiento de los componentes, estos escenarios serán puestos a prueba con la herramienta SIPp debido a que permite la creación de distintos escenarios de prueba y la generación de alta concurrencia en el establecimiento de sesiones mediante el protocolo de señalización SIP.

Palabras Claves: Arquitectura, Contingencia, Balanceo de Carga, Clustering, SIP, Kamailio, Asterisk

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

El nacimiento de nuevas tecnologías en el campo de las comunicaciones mediante la utilización de redes de computadoras ha impulsado el crecimiento de las organizaciones, las cuales, interesadas en la utilización eficiente de recursos, apuestan a mecanismos innovadores de comunicación, estos permiten el establecimiento de conexiones entre dispositivos para el intercambio de información, permitiendo la comunicación de entes bien identificados. Habitualmente estas tecnologías de comunicación siguen un esquema de funcionamiento bien normado, los cuales definen de manera clara puntos intermedios, estados, procesos y posibles comportamientos de la comunicación, estos son denominados protocolos de comunicación.

Extrapolando entonces dichos conceptos tenemos que la existencia de organizaciones interesadas en diversificar sus capacidades de comunicación de forma óptima se inclinan por la utilización de estas tecnologías y protocolos de comunicación de red; sin embargo existen casos en los que la magnitud de estas organizaciones requiere de infraestructuras de red que prioricen el rendimiento y la estabilidad de la plataforma de comunicación y es en este punto donde surgen varias interrogantes:

- ¿Qué protocolo es el más adecuado?
- ¿Qué infraestructura de red es la más óptima a nivel de rendimiento?
- ¿Qué infraestructura de red es la más óptima a nivel de durabilidad y escalabilidad?

Para dar respuesta a estas interrogantes surge el siguiente documento, el cual parte de la utilización de un protocolo de comunicación de red mediante la tecnología VoIP (Voice over IP) como el protocolo SIP (Session Initiation Protocol) [1], y la utilización de herramientas que permitan la instauración de una infraestructura de red de alta disponibilidad y buen desempeño, tales como Kamailio, CoroSync, Asterisk, entre otros.

A lo largo del desarrollo del siguiente trabajo de investigación se abordan un total de seis capítulos principales, los cuales dan respuesta a las interrogantes antes planteadas. A continuación se da una breve descripción del contenido de los mismos, lo cual se pretende sea un medio que facilite al lector la comprensión de la distribución del contenido del documento:

Capítulo 1 - El Problema: Contiene el razonamiento detrás de la ubicación del problema y la definición de los objetivos que permitirán tener la solución a dichos problemas

Capítulo 2 - SIP (Session Initiation Protocol): Tiene como finalidad dar una introducción al lector sobre los procesos de comunicación establecidos en el protocolo SIP, así como permitir tener una visión general de todo aspecto necesario a para la generación de un perfil técnico del protocolo de comunicación.

Capítulo 3 - Mecanismos de Alta Disponibilidad: Para la creación de una infraestructura de red que garantice los estándares de rendimiento establecidos por las organizaciones es necesario que se consoliden aquellos mecanismos que permiten que el servicio prestado de comunicación esté disponible de manera segura y permanente. Este capítulo se orienta en ahondar los conceptos necesarios para el entendimiento de mecanismos existentes para permitir a la infraestructura planteada estar disponible ante distintos contingencias probables dentro de la misma.

Capítulo 4 - Balanceo de Carga: Para el manejo de grandes volúmenes de datos, las prácticas habituales del pensamiento centralizado no resultan de mucha utilidad, por tanto es común que ver estos escenarios la colocación de múltiples nodos de procesamiento. En este escenario se requiere de algún mecanismo que haga llegar la carga de trabajo a todos estos nodos de manera óptima, es decir la que más se ajuste a las características de la infraestructura de red, es allí donde se ahonda en los conceptos base relativos a las tecnologías y métodos de balanceo de carga.

Capítulo 5 - Herramientas de Interés: La utilización de múltiples herramientas dirige el flujo de la investigación hacia la definición de las mismas y de sus conceptos base, siendo este el punto focal de este capítulo, la documentación de todas estas tecnologías y herramientas que permitirán la creación de la infraestructura deseada.

Capítulo 6 - Trabajos Relacionados: La experiencia recabada en trabajos afines, sirve de apoyo para la creación de nuevas teorías y nuevos productos, por ello que este capítulo hace referencia a algunos trabajos de interés que posean una relación directa con la investigación realizada, brindando robustez y soporte a los conceptos y técnicas utilizadas para el desarrollo de la solución.

Capítulo 7 - Marco Metodológico: Describe los procedimientos que se plantean para la implementación de la arquitectura de VoIP deseada.

Capítulo 8 - Arquitectura de Alta Disponibilidad de Telefonía VoIP: Contiene el diseño de la arquitectura realizada, la descripción de las actividades realizadas para lograr dicho objetivo y el resultado de la misma.

Capítulo 9 - Escenarios de Prueba y Comportamientos Esperados: En base al resultado de la arquitectura implementada, se definen los escenarios de prueba y el comportamiento esperado en ámbitos de contingencia y alta concurrencia.

Capítulo 10 – Verificación y Análisis de Resultados: Comprehende el proceso de análisis de los resultados obtenidos en los escenarios de prueba basándose en los comportamientos esperados descritos en el capítulo anterior.

2. El Problema

A lo largo de este capítulo se llevará a cabo la descripción y planteamiento del problema desde distintas perspectivas, la justificación por la cual los problemas, en cuanto a mecanismos de contingencia y escalabilidad en soluciones de telefonía VoIP requieren una solución óptima, y la propuesta de implementación para lograr atacar estos problemas. Para definir una propuesta bien elaborada, es importante definir los objetivos de ésta como se podrá ver a lo largo de este capítulo.

2.1 Planteamiento Del Problema

Desde la perspectiva tecnológica se han identificado ciertas limitaciones de determinados planteamientos de arquitecturas de telefonía IP orientadas a la prestación de un servicio de telefonía de calidad, particularmente en entornos de mucha concurrencia donde se describe una carga de trabajo significativa como lo son entornos organizacionales, en los cuales un conjunto de usuarios consumen servicios de manera sostenida y reiterativa. Estas limitaciones pueden llegar a significar un punto de quiebre en el cual una organización, en aras de la mejora de calidad de servicio y la no interrupción del mismo, puede optar por soluciones propietarias que en muchos casos representan una inversión monetaria significativa. Es entonces posible afirmar que el principal problema que presentan las organizaciones que emplean soluciones de software libre para telefonía son aquellas que impiden la continuidad del servicio en entornos de elevada concurrencia y alta demanda.

Muchos de estos problemas se deben a que determinadas organizaciones ignoran los parámetros de exigencia que se podrían esperar en los ambientes a los que se ve sujeto el servicio en su organización. Por ende, dichas organizaciones no evalúan ciertas soluciones topológicas y de arquitectura con herramientas de software libres disponibles para atacar inconvenientes, los cuales podrían resolverse ofreciendo mecanismos proactivos de alta disponibilidad y de contingencia en caso del mal funcionamiento del sistema por distintas razones. Además, si se prevé el posible crecimiento en las exigencias de los servicios gestionados por la organización, un buen planteo en la gestión y configuración de los servicios a ofrecer dentro de la arquitectura, más soluciones de balanceo de carga, permitirían una mayor escalabilidad en el servicio.

2.2 Justificación Del Problema

Permitir la comunicación eficaz de dos entes, es uno de los objetivos primordiales del área de las comunicaciones, en particular en el área de las redes computacionales. El objetivo más claro es permitir que una red sea empleada para el intercambio efectivo de datos, lo que representa un componente imprescindible para las organizaciones, ya que éstas hacen uso constante de las redes de computadores para afianzar entornos laborales y para la realización de tareas cotidianas.

En algunos casos estas organizaciones describen un volumen de usuarios significativo, en tal grado que la presencia de este conglomerado activo en la red de comunicación pone en riesgo el funcionamiento de la misma. Es aquí donde surge la necesidad perentoria del diseño de una arquitectura que permita disminuir los riesgos de

inoperatividad del sistema de comunicación causado por elevados niveles de concurrencia utilizando herramientas de software libre existentes.

El concepto de alta disponibilidad y continuidad de la operación en redes de computadores viene a significar una solución para entornos de alta carga de trabajo. Sin embargo, el costo asociado a la implementación con soluciones propietarias de estas medidas de mejora de la calidad del servicio de comunicación hace que estas soluciones sean inviables para algunas organizaciones. Y es en este punto donde surge el interés de realizar una arquitectura que permita establecer parámetros de alta disponibilidad mediante el uso de herramientas de software libre. Ello, teniendo en cuenta que las limitaciones particulares de cada herramienta puede verse opacada con la concurrencia de múltiples herramientas sincronizadas entre sí para ofrecer un servicio de calidad, haciendo mención de que en la unión y aprovechamiento de las ventajas particulares está la fuerza.

2.3 Objetivos

A continuación se describen los objetivos que se desean lograr con la realización del trabajo propuesto.

2.3.1 Objetivo General

Proponer una arquitectura que permita el establecimiento de la comunicación mediante el uso de la tecnología VoIP, específicamente usando el protocolo SIP, que garantice la alta disponibilidad, escalabilidad y el correcto funcionamiento del servicio de telefonía.

2.3.2 Objetivos Específicos

- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores SIP Asterisk para dar recomendaciones sobre la configuración de la arquitectura a implementar
- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores SIP Kamailio para dar recomendaciones sobre la configuración de la arquitectura a implementar
- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores con Pacemaker para dar recomendaciones sobre la configuración de la arquitectura a implementar
- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores con Corosync para dar recomendaciones sobre la configuración de la arquitectura a implementar
- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores con DRBD o almacenamientos compartidos para dar recomendaciones sobre la configuración de la arquitectura a implementar
- Realizar estudios sobre la instalación, operación, configuración y mantenimiento de servidores MySQL para dar recomendaciones sobre la configuración de la arquitectura a implementar

- Realización de escenarios de pruebas para mecanismos de alta disponibilidad y de contingencias para la verificación de su funcionamiento y obtener resultados esperados en el restablecimiento del servicio
- Análisis de medición del protocolo SIP y realización de pruebas de estrés mediante herramientas como SIPp para la obtención de resultados, que permitan establecer medidas comparativas de rendimiento

2.4 Alcances

Lograr la construcción de una infraestructura de telefonía IP que esté en la capacidad de garantizar la prestación del servicio ante la posibilidad de un mal funcionamiento del sistema y presencia de picos de flujos de comunicación pronunciados en ciertos ambientes.

De ser necesario recurrir a mecanismos de failover (sea por fallas en los servicios provistos o de hardware), realizar una configuración que permita establecer mecanismos de control de flujo; esto dependiendo de la carga que se estime deba ser soportada por la solución.

3. SIP: Session Initiation Protocol

Dentro del conjunto de tecnologías existentes en el mundo de las telecomunicaciones existe la necesidad de establecer mecanismos que permitan iniciar un intercambio de información entre dos actores bien definidos. Para ello la comunidad informática y organizaciones dedicadas a la estandarización dedican tiempo y esfuerzo a tareas de documentación que permitan que cualquier ente, mediante el cumplimiento de un conjunto de protocolos y premisas, pueda establecer un canal de comunicación con un ente par. El protocolo SIP (Session Initiation Protocol) [1] cumple con estas funciones específicas, siendo un protocolo orientado a la estandarización de mecanismos de envío y recepción de mensajes de presencia así como envío y recepción de mensajes instantáneos. Este comportamiento nos dice que SIP como protocolo red se desempeña y aprovecha la infraestructura de red, en la cual el punto focal es la comunicación de endpoints mediante la inicialización de sesiones de comunicación.

3.1 Visión General del Protocolo SIP

Como su nombre lo indica, SIP (Session Initiation Protocol) [1] es un protocolo orientado al establecimiento de sesiones de comunicación, es decir permite a dos endpoints o entes, como comúnmente serán llamados durante el desarrollo del siguiente documento a los elementos extremos del proceso de comunicación, la apertura de un canal de comunicación. Dentro de los flujos normales de operación tenemos los descritos por la Figura 3.1, en la cual podemos ver el conjunto de operaciones básicas que son llevadas a cabo por el protocolo de acuerdo por la especificación técnica suministradas por el RFC 3261.

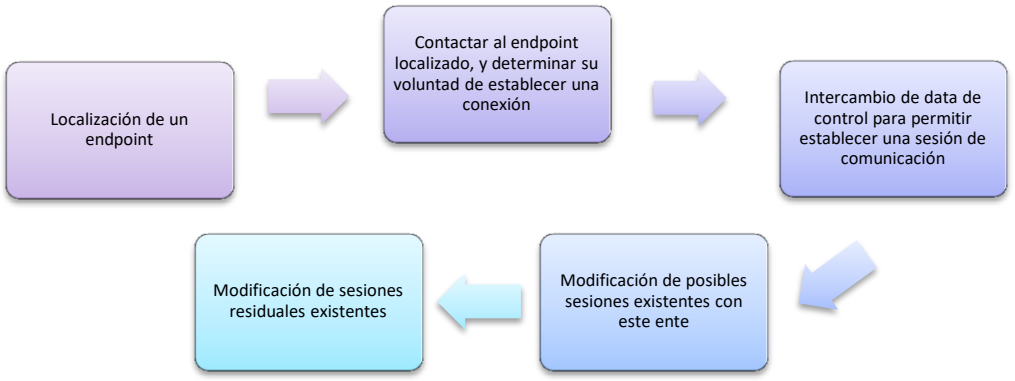


Figura 3.1: Flujo de Funcionamiento del Protocolo SIP

El flujo de operación inicia como un proceso de descubrimiento de un candidato para la creación de una sesión, por ser un protocolo de capa de aplicación, SIP se apoya en múltiples funciones especificadas de la pila de protocolos TCP/IP (Transmission Control Protocol/Internet Protocol) y UDP/IP (User Datagram Protocol/Internet Protocol). La selección de un candidato óptimo para la apertura de la comunicación es un proceso de

validación e identificación de ambos extremos que permite conocer la identidad de un ente solicitante de apertura de sesión a nivel interno del protocolo. Posteriormente el proceso de intercambio de datos de control permite la identificación de las necesidades de apertura de conexiones por parte de los entes involucrados, esto debido a que es necesario confirmar la necesidad de los entes involucrados para dar inicio a una sesión. Debido a que el protocolo SIP establece que una sesión existente puede poseer un conjunto de entes y que por tanto esta sesión previamente establecida puede ser modificada para dar cabida a un nuevo integrante de sesión, se procede a consultar las operaciones a realizarse en la sesión lo cual se conoce como un proceso modificación de sesión.

El último, pero no menos importante, flujo de operación del protocolo consiste en una etapa de contingencia y evaluación de las necesidades de modificación de una sesión SIP, la misma puede consistir en la eliminación, transferencia y modificación de los parámetros de comunicación e invocación de primitivas disponibles. Este flujo brinda servicios necesarios para permitir la adaptabilidad del protocolo, haciendo que la comunicación sea estable y flexible [1].

3.2 Visión de Funcionamiento del Protocolo SIP

Para entender el funcionamiento del protocolo SIP nos apoyamos en una serie de ejemplos que pueden servir como punto referencial para la definición de los conceptos primordiales que influyen el protocolo.

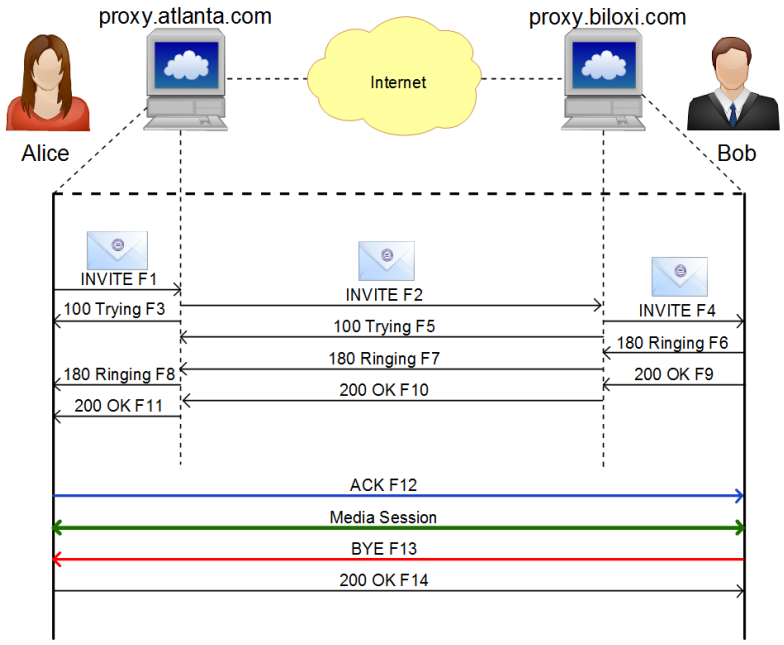


Figura 3.2: Llamada SIP Utilizando Proxy Servers

En la Figura 3.2 podemos ver el proceso de comunicación entre dos usuarios. Alice y Bob, a través de sus dispositivos de comunicación SIP sobre la infraestructura del Internet, cada mensaje está marcado con la letra “F” y un número indicativo del orden

de precedencia del evento dentro del mapa de tiempo [1].

Alice realiza una llamada a Bob utilizando para ello el identificador SIP asignado a Bob, sip:bob@proxy.biloxi.com, que consiste en un URI (Uniform Resource Identifier) llamado SIP URI. Estos identificadores son definidos más adelante en el desarrollo de este documento. SIP está basado en un mecanismo de solicitud y respuesta similar al

```
1: INVITE sip:bob@proxy.biloxi.com SIP/2.0
2: Via: SIP/2.0/UDP pc33.proxy.atlanta.com;branch=z9hG4bK776asdhds
3: Max-Forwards: 70
4: To: Bob <sip:bob@proxy.biloxi.com>
5: From: Alice <sip:alice@proxy.atlanta.com>;tag=1928301774
6: Call-ID: a84b4c76e66710@pc33.proxy.atlanta.com
7: CSeq: 314159 INVITE
8: Contact: <sip:alice@pc33.proxy.atlanta.com>
9: Content-Type: application/sdp
10: Content-Length: 142
```

Figura 3.3: Cabecera de Ejemplo para Apertura de Conexión

existente en el protocolo HTTP, cada transacción consiste de una solicitud y al menos una respuesta por parte del servidor. En el ejemplo antes expuesto en la Figura 3.2, podemos ver como el proceso empieza por el envío de una solicitud INVITE por parte de Alice dirigida a Bob. Invite es un ejemplo de un acción que un emisor (Alice) solicita a un receptor (Bob) realizar [1]. El cuerpo del mensaje INVITE contiene una serie de campos que proveen de información sobre el mensaje. Un ejemplo de la cabecera SIP que se emplea en el flujo de comunicación puede ser visto en la Figura 3.4, la estructura y elementos que componen esta serán definidos en las próximas secciones de este documento [1].

Debido a que el dispositivo SIP de Alice no posee información sobre la ubicación de Bob, o del dominio proxy.biloxi.com, envía la solicitud INVITE al punto de acceso más cercano, en este el servidor proxy.atlanta.com. La dirección de red de dicho servidor puede haber sido configurada en el dispositivo de Alice o ser parte de las configuraciones adicionales recibidas en el proceso de configuración DHCP (Dynamic Host Configuration Protocol) [1].

Los servidores proxy.atlanta.com y proxy.biloxi.com son conocidos como SIP Proxy, los mismos realizan funciones de reenvío y solicitud al servicio de solicitantes. En el ejemplo expuesto, el servidor proxy.atlanta.com recibe una solicitud INVITE proveniente del dispositivo de Alice y responde con un mensaje Trying (100). Esta respuesta indica al solicitante que la solicitud ha sido recibida y que el servidor proxy cumple funciones de enrutamiento. Las respuestas en el protocolo SIP emplean un número de tres dígitos seguidos por frases descriptivas del evento a comunicar. Seguidamente el servidor proxy proxy.atlanta.com localiza a su entidad par, proxy.biloxi.com, posiblemente realizando un conjunto de solicitudes mediante el protocolo DNS (Domain Name System), y realiza el reenvío de la solicitud SIP. Al momento de realizar este envío, se enmascara el mensaje con un campo adicional en

la cabecera SIP que permite que el servidor destino pueda enviar una respuesta de vuelta sin la necesidad de realizar consultas DNS.

Posteriormente el servidor proxy.biloxi.com responde a la solicitud con un mensaje Trying (100) notificando que está activo y que realizará las tareas de enrutamiento para el establecimiento de la conexión. El servidor proxy consulta generalmente a una base de datos, llamada servicio de localización, la cual contiene la dirección de red actual de Bob.

El servidor proxy.biloxi.com agrega otro campo a la cabecera para permitir el enrutamiento de vuelta desde al dispositivo SIP de Bob.

El dispositivo de Bob recibe el mensaje INVITE y alerta de la llamada entrante proveniente de Alice, para ello emplea la información de la cabecera SIP. El dispositivo de Bob genera un tono de llamada entrante, lo cual origina una respuesta Ringing (180) hacia Alice, notificando que la llamada está en espera de atención. Este mensaje es enrutado empleando la información de la cabecera del mensaje que ha sido colocada por los dispositivos intermedios. Cuando el dispositivo SIP de Alice recibe la información de que la llamada está en espera de respuesta en el dispositivo de Bob, se genera un tono de repique.

Bob contesta la llamada, en ese instante se envía un mensaje de vuelta a Alice, OK (200) conjuntamente con información sobre qué tipo de sesión se está dispuesto a establecer, ambos extremos de la comunicación poseen la capacidad de determinar qué sesión será establecida. Finalmente el dispositivo de Alice envía un mensaje ACK dirigido directamente a Bob, ya que, gracias a los mensajes previos de negociación, las direcciones de red de ambos entes son bien conocidas entre sí. El dispositivo de Alice detiene el sonido de repique. Los servidores proxy intermedios que realizaban búsquedas continuas abandonan el flujo de llamada, esto marca el final del proceso de establecimiento de conexiones, lo que es denominado INVITE/200/ACK three-way handshake [1].

En el transcurso de la sesión ambos extremos de la comunicación pueden desear realizar la modificación de las características de la sesión establecida. Esto genera el reenvío de un mensaje re-INVITE, el cual contiene una descripción de los datos a ser modificados. Este mensaje re-INVITE viaja utilizando la sesión establecida, en caso de que las nuevas configuraciones sean aceptadas por el ente par, se envía un mensaje OK (200) para confirmar este cambio el cual es seguido de un mensaje ACK. En caso de que el cambio no sea aceptado por un ente, este envía un mensaje Not Acceptable Here (488), el cual también recibe un mensaje de confirmación de recepción.

Como punto final Bob cuelga la llamada, lo cual genera el envío de un mensaje BYE, que es enrutado directamente hasta Alice, quien confirma la recepción del mensaje de finalización de sesión, con un mensaje OK (200). No se realiza el envío de un mensaje ACK para este último, ya que solo se realiza el envío de un mensaje de confirmación

como respuesta a una respuesta de un mensaje INVITE. Por esta razón el manejo de solicitudes en SIP es clasificado entre INVITE y no INVITE, refiriéndose a cualquier otro método.

En algunos casos puede ser de utilidad para los servidores intermedios, entiéndase servidores proxy para el ejemplo planteado en la topología descrita en la Figura 3.3, visualizar todo el tráfico durante la duración de la sesión, en este caso deben agregar un campo Record Route a la cabecera del mensaje inicial de invitación al inicio de sesión, INVITE, el cual contiene el URI resuelto para el dispositivo o dirección IP del servidor proxy. Esta información es recibida por los entes de la comunicación, los cuales al detectar la existencia del campo Record Route realizan el almacenamiento de la ruta al proxy al cual deben realizar el envío de los mensajes intercambiados en el desarrollo de la sesión. Cada servidor intermedio puede decidir a voluntad si desea recibir los mensajes generados por una sesión SIP, los cuales serán pasados de un proxy a otro de manera recursiva, dependiendo siempre de la voluntad del mismo de recibirlos.

El registro es otra operación muy común en SIP, la cual consiste en un método de aprendizaje de datos de localización actual por parte de los servidores. A través del proceso de inicialización de la comunicación y cada cierto tiempo, los dispositivos envían mensajes REGISTER a los servidores SIP, conocidos como SIP registrar. Estos mensajes realizan la asociación de un usuario con el dispositivo desde el cual este ha iniciado sesión para ese instante de tiempo. Esta asociación es llamada “binding” o enlace, y se realiza mediante el registro en una base de datos de información de localización, denominada servicio de localización, sin ser propiamente un servicio. Muy a menudo los servidores registrar son localizados conjuntamente con el servidor proxy.

3.3 Mensajes SIP

SIP es un protocolo basado enteramente en texto, el cual utiliza el conjunto de caracteres estándar UTF-8, por lo que la estructura de los mensajes de intercambio puede ser definida fácilmente. Tanto los mensajes de solicitud como de respuesta utilizan un formato básico definido en el RFC 2822 [2] para mensaje de transferencia en el Internet, incluso cuando la sintaxis difiere, SIP permite múltiples campos de cabecera. Esto brinda a la estructura del mensaje cierto nivel de flexibilidad, tal y como puede ser visto en la Figura 3.6. En la misma también puede verse como se incluye un campo denominado CRLF, que tiene como finalidad marcar la frontera entre el cuerpo del mensaje SIP y el conjunto de cabeceras de diferente tipo.

Este campo CRLF, está definido en el RFC 2822 [2] como un campo de división, y consiste de CR (Carriage Return) y LF (Line Feed), juntos componen lo que normalmente se conoce como el carácter nueva línea [1].

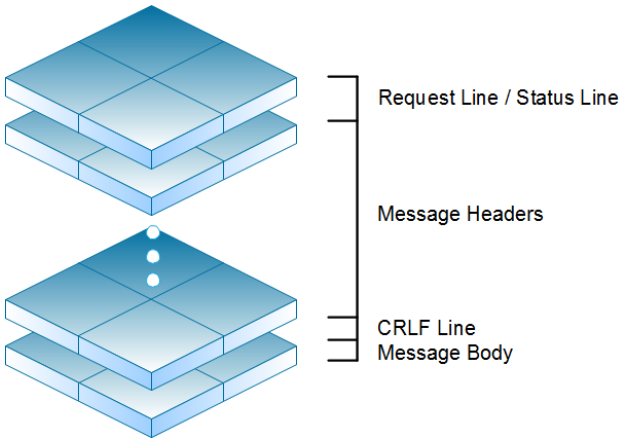


Figura 3.4: Representación Gráfica de Mensajes SIP

3.3.1 Request Message

Una Solicitud SIP es diferenciada por su línea inicial o Request Line, tal y como se refleja en la Figura 3.4. La estructura definida por el RFC 3261 [1] dicta que se debe tener una línea de texto, con campos separados por el carácter espacio (SP) conformada por:

- Method (Método): Método a ser consulado a un servidor o ente par sobre sus capacidades.
- Request-URI (URI de Solicitud): Consiste en un URI SIP, este indica el usuario o servicio al cual está orientada esta solicitud.
- SIP Version: Ambos mensajes, solicitud y respuesta, deben poseer la misma versión del protocolo SIP para permitir un nivel máximo de acoplamiento [1].

Dando esto como resultado una línea equivalente a la mostrada en la Figura 3.5.

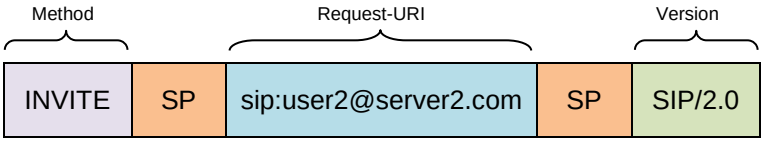


Figura 3.5: Representación Gráfica de una Solicitud SIP

3.3.1.1 Métodos Existentes

Dentro del alcance del RFC 3261 [1] podemos encontrar un conjunto de métodos descritos, para el establecimiento de conexiones SIP, los mismos son listados y definidos en la Tabla 3.1 [1].

Método	Definición
INVITE	Empleado para el establecimiento de sesiones de intercambio de contenido entre dos agentes. Usualmente el cuerpo de este mensaje contiene la información del ente que realiza la llamada.
REGISTER	Utilizado por un agente para notificar a una red SIP de su URI de contacto actual (Dirección IP) y la información de localización dada por el URI. Este método es el disparador de los eventos de “binding” explicados en la Sección 2.3 de este mismo documento.
BYE	Usado por un ente para terminar una sesión establecida. Una sesión es considerada como establecida si una solicitud INVITE ha recibido una respuesta satisfactoria o en su defecto una respuesta ACK.
ACK	Conocido como mensaje de confirmación final para solicitudes con el método INVITE. En general cualquier respuesta satisfactoria recibe un mensaje ACK.
CANCEL	Como visto en la Sección 2.3, este método es empleado para terminar una sesión establecida, previniendo respuestas a futuras solicitudes del tipo INVITE. Puede ser generada por un agente o servidores intermedios como servidores proxy y servidores registrar.
OPTIONS	Método empleado para consultar a un agente las características de soporte de una posible conexión a ser establecida, adicionalmente puede realizar el descubrimiento de las características de una sesión ya establecida.

Tabla 3.1: Métodos Definidos por el RFC 3261

Adicionalmente existe un conjunto de métodos que han sido definidos en documentos estándar separados, los mismos poseen relevancia, sin embargo, su existencia en el protocolo depende enteramente de la implementación del mismo. A continuación se realiza la definición de estos métodos agregados en la Tabla 3.2 [3] [4] [5] [6].

Método	RFC	Definición
SUSCRIBE	RFC 6665	Empleado para realizar o crear una solicitud de estado actual de un nodo de la red SIP desde un nodo remoto.
NOTIFY	RFC 6665	Solicitudes enviadas producto de una suscripción, en la misma se envía a los suscriptores información sobre cambios de estado de un ente.
PUBLISH	RFC 3903	Permite realizar la publicación, modificación y remoción, ante la red SIP, de estado de eventos. Esta funcionalidad puede ser útil en infraestructuras SIP donde se haga uso de un Presence Server (Servidores de Presencia).
REFER	RFC 3515	Usado para indicar a un agente que debe realizar la solicitud a otro agente para acceder a un recurso, utilizando para ello su URI o URL. El comportamiento de este método puede ser comparado con una llamada a procedimiento remoto o RPC.
UPDATE	RFC 3311	Este método se emplea para realizar modificaciones al estado de una sesión sin cambiar el estado del dialogo, con el objetivo de actualizar el estado del mismo. Algunos escenarios donde se puede ver la utilidad de este método son la colocación de llamadas en espera (Hold State) o sin emisión de sonido (Mute).

Tabla 3.2: Conjunto de Métodos Adicionales Registrado para el Protocolo SIP

3.3.1.2 Request URI

El nombre URI (Uniform Resource Indicators) hace referencia a una cadena de caracteres que identifica de forma univoca a un recurso disponible en los procesos de comunicación. En general todo URI, SIP URI y SIPS URI están almacenados bajo dominios web. Un SIP Request URI contiene suficiente información para indicar y mantener una sesión de comunicación con un recurso, algunos ejemplos de recursos son los siguientes [1]:

- Servicio de usuario.
- Una apariencia en un teléfono multilinea.
- Un buzón de mensajes en un servicio de mensajería.
- Un número PSTN (Public Switched Telephone Network) en un servicio de

acceso o Gateway Service.
- Un grupo en una organización.

Componentes de un SIP URI

$$(sip|sips)[:user[:password]@]host[:port][\{;pname=pvalue\}][\{?hname=hvalue\}]$$

Figura 3.6: Forma EBNF de un Request URI

Los componentes principales de un URI descritos en la Figura 3.6. En la misma podemos ver la existencia de elementos clave que conforman los identificadores definidos en la Tabla 3.3.

3.3.2 Response/Reply Message

Las respuestas SIP son diferenciadas de las solicitudes debido a que estas poseen la línea Status Line como línea inicial del cuerpo del mensaje.

Una línea de estado o Status Line consiste de [1]:

- SIP Version: versión del protocolo SIP a utilizar
- Status Code: número de tres dígitos representativo del evento a informar
- Reason Phrase: texto breve explicativo del evento acontecido

Esta distribución puede ser vista en la Figura 3.7 [1].

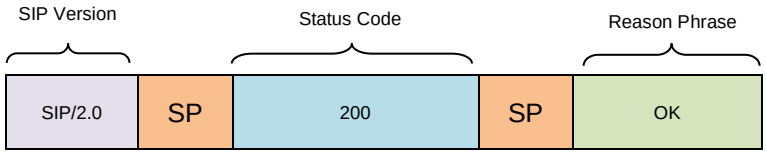


Figura 3.7: Representación Gráfica de una Respuesta SIP

Campo	Definición	Valor de Ejemplo
Esquema (scheme)	Esquema de comunicación a ser empleado, SIP o SIPS.	sip: sips:
Usuario (user)	Identificador de un recurso a ser accedido en el host especificado	alice bob 117
Contraseña (password)	Contraseña de acceso asociada al usuario suministrado	123456 S3cR3t $3©®€t
Host de Conexión (host)	Host que provee el recurso para su consumo, en este caso puede ser un nombre de dominio o una dirección de red.	Sipserver.biloxy.com
Puerto de Conexión (port)	Puerto de conexión para el acceso al recurso en el host especificado.	5060 5061
Parámetros (pname, pvalue)	• **transport**, determina el mecanismo de transporte, bien sea TCP o UDP. • **maddr**, dirección física del servidor a ser contactado en busca del usuario especificado. • **ttl**, valor establecido para la métrica de time-to-live de un paquete. Solo debe ser utilizado en esquemas de multicast, cuando la dirección especificada por maddr sea una dirección multicast. • **user,** su existencia se basa en permitir diferenciar entre nombres de usuario y números telefónicos que pueden ser tomados por su forma numérica como identificadores de llamada. • **method,** el método de una solicitud SIP puede ser especificado también en este campo. • **lr,** campo empleado para especificar que el elemento responsable por el recurso a solicitar maneja o implementa mecanismos de enrutamiento. Tal y como sucede con el envío de solicitudes de Record-Route.	sip:+1-212-555-1212:1234@gateway.com;user=phone
Cabeceras (hname, hvalue)	Campos acceso a recurso, separados del identificador de usuario y puerto por el carácter “?”. A su vez son separados por el carácter “&” entre si.	sips:alice@atlanta.com?subject=project%20x&priority=urgent

Tabla 3.3: Componente de un SIP URI

3.3.2.1 Códigos de Respuesta

Tipo de Respuesta	Identificador	Frase Explicativa
Estado provisional de la comunicación	100	Trying – Intentando
	180	Ringing – Sonando
	181	Call Is Being Forwarded – Llamada Está Siendo Transferida
	182	Queued – Encolada
	183	Session Progress – Llamada en Progreso
Éxito de la comunicación	200	OK – OK
	202	Accepted – Aceptada
Reenvío necesario de la petición SIP	300	Multiple Choices – Múltiples Opciones
	301	Moved Permanently – Movido Permanentemente
	302	Moved Temporarily – Movido temporalmente
	305	Use Proxy – Usar Proxy
	380	Alternative Service – Servicio Alternativo
Errores del cliente	400	Bad Request – Mala Petición
	401	Unauthorized – No Autorizado
	402	Payment Required – Se Requiere Pago
	403	Forbidden – Prohibido
	404	Not Found – No Encontrado
	405	Method Not Allowed – Método no Permitido
	406	Not Acceptable – No es Aceptable
	407	Proxy Authentication Required – Se Requiere Autenticación
	408	Request Timeout – Tiempo Agotado para la Petición
	410	Gone – Ya no Existe
	413	Request Entity Too Large – Petición Demasiado Grande
	414	Request URI Too Long – URI Demasiado Largo
	415	Unsupported Media Type – Tipo de Medio no Soportado
	416	Unsupported URI Scheme – Esquema URI no Soportado
	420	Bad Extension – Extensión Incorrecta
	421	Extension Required – Se Requiere Extensión
	423	Interval Too Brief – Intervalo Demasiado Corto
	480	Temporarily Unavailable – No Disponible Temporalmente
	481	Call/Transaction Does Not Exist – No Existe la Llamada/Transacción
	482	Loop Detected – Bucle Detectado
	483	Too Many Hops – Demasiados Saltos
	484	Address Incomplete – Dirección Incompleta
	485	Ambiguous – Ambiguo
	486	Busy Here – Ocupado
	487	Request Terminated – Petición Terminada
	488	Not Acceptable Here – No es aceptable Aquí
	491	Request Pending – Petición Pendiente
	493	Undecipherable – Indescifrable
Informan de errores del servidor	500	Server Internal Error – Error Interno del Servidor
	501	Not Implemented – No Implementado
	502	Bad Gateway – Pasarela (Gateway) Equivocada
	503	Service Unavailable – Servicio no Disponible
	504	Server Time-out – Tiempo Agotado en el Servidor
	505	Version Not Supported – Versión no Soportada
	513	Message Too Large – Mensaje Demasiado Largo
Informan de errores generales	600	Busy Everywhere – Ocupado en todos Sitios
	603	Decline – Rechazado
	604	Does Not Exist Anywhere – No Existe en nigún Sitio
	606	Not Acceptable – No Aceptable

Tabla 3.4: Listado de Códigos de Respuesta con su Correspondiente Frase Explicativa

3.3.3 Definición de Cabecera de Mensajes

Las cabeceras del mensaje SIP vienen a continuación de la línea inicial, Start Line, la cual como hemos visto en secciones anteriores puede ser empleada como Request o Status Line, las mismas proveen información sobre la solicitud o la respuesta según sea el caso [1]. Una cabecera de mensajes SIP consiste de:

- Nombre de la Cabecera, seguido del carácter ":" (dos puntos).
- Valor de la Cabecera.

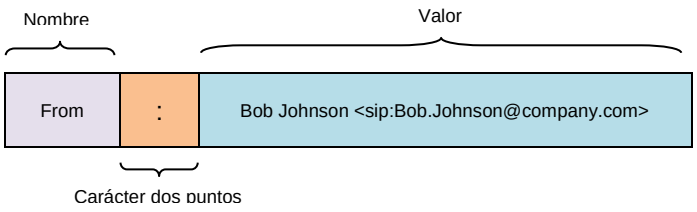


Figura 3.8: Cabecera From de Mensaje SIP

Adicionalmente en esta misma figura se puede notar que una cabecera de mensajes puede contener múltiples campos, como el caso de la cabecera SIP que posee el nombre del usuario y su URI respectivo. Las cabeceras de mensajes definidas dentro del RFC 3261 [1] son presentadas en la Figura 3.9 [7].

Accept	**Content-language**	**Organization**	**Supported**	**Content-disposition**
Accept-encoding	**Content-length**	**Priority**	**Timestamp**	**Max-forwards**
Accept-language	**Content-type**	**Proxy-authenticate**	**To**	**MIME-version**
Alert-info	**Cseq**	**Proxy-authorization**	**Unsupported**	**Content-encoding**
Allow	**Date**	**Proxy-require**	**User-agent**	**Contact**
Also	**Encryption**	**Record-route**	**Via**	**In-reply-to**
Authorization	**Error-info**	**Require**	**Warning**	**Server**
Call-ID	**Expires**	**Response-key**	**WWW-authenticate**	**Subject**
Call-info	**From**		**Route**	**Retry-after**

Figura 3.9: Conjunto de Cabeceras Definidas en el RFC 3261

3.3.3.1 Campos más Relevantes de la Cabecera SIP

A continuación se da una descripción de aquellas cabeceras de mensaje que son consideradas como las más relevantes para el lector.

Call-ID

Representa una relación compartida entre dos usuario SIP, identifica una invitación

particular a realizar un intercambio de transacciones y a estas propiamente. En un proceso de comunicación se asocia este identificador para ayudar a asociar eventos de llamada con una sesión determinada [1]. Un ejemplo de esta puede ser visto en la Figura 3.10.

Figura 3.10: Cabecera Call-ID de Mensaje SIP

Contact

Provee un URL bajo el cual un usuario puede ser alcanzado directamente, es decir sin interferencia de algún servidor intermedio. Esta funcionalidad resulta importante a la hora de realizar las tareas de enrutamiento para servidores de registro SIP, Registrar Server, ya que permite que un usuario sea alcanzable directamente, prescindiendo así del contacto con servidor de registro [1]. Un ejemplo de esta puede ser visto en la Figura 3.11.

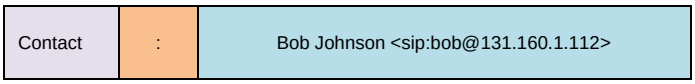


Figura 3.11: Cabecera Contact de Mensaje SIP

Cseq

La cabecera Cseq (Command Sequence) posee dos campos:

- Un número entero, utilizado para realizar una solicitud de un método dentro de la misma sesión.
- Un método, método a invocar dentro del mensaje.

El uso de este campo de la cabecera permite la modificación de datos dentro de una sesión ya establecida, adicionalmente el número Cseq permite identificar o trazar un flujo de funcionamiento del protocolo dando una jerarquía de precedencia transaccional [1]. Un Ejemplo de esta puede ser visto en la Figura 3.12.

Figura 3.12: Cabecera Cseq del Mensaje SIP

From

Contiene la identificación del ente que inicia el proceso de comunicación mediante SIP, este posee la información para poder realizar una conexión directa al usuario emisor de la solicitud INVITE [1]. Un Ejemplo de esta puede ser visto en la Figura 3.13.

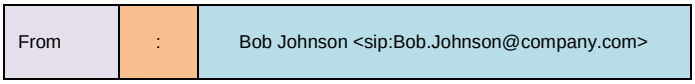


Figura 3.13: Cabecera From del Mensaje SIP

Record-Route y Route

Empleado por elementos intermedios del proceso de comunicación para establecer un canal de notificación sobre el flujo de información que en el mismo se desarrolla durante el periodo de existencia de una sesión. Esta necesidad de permanecer en el camino de comunicación de manera permanente puede estar motivada por diferentes causales, entre los que tenemos políticas de seguridad de un sitio seguro en el cual sea necesario establecer un perímetro de control de acceso para peticiones entrantes.

Este mecanismo de agregación de cabeceras puede ser visto como una estructura de datos del tipo Pila, en la que se realizan una inserción a medida que un mensaje es capturado por un ente que desee permanecer a la escucha de los paquetes durante el proceso de enrutamiento que se desarrolla en una sesión [1]. Un Ejemplo de esta puede ser visto en la Figura 3.14.

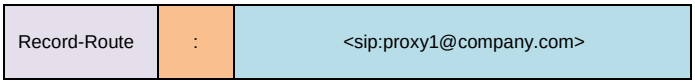


Figura 3.14: Cabecera Record-Route del Mensaje SIP

To

Esta cabecera del mensaje SIP siempre contiene el ente designado como receptor del paquete en transporte, en algunas ocasiones posee una dirección de red e lugar de una dirección de nombre. En este punto se debe hacer una distinción importante entre este campo de cabecera de mensajes y el URI suministrado en la Línea Inicial, Request Line de una solicitud SIP. En este sentido el valor de la cabecera no es modificado en el desarrollo del proceso de comunicación, a diferencia del Request-URI el cual posee la dirección del próximo salto a tomar dentro del camino de enrutamiento al destinatario y por consecuencia es modificado en cada salto [1]. Un Ejemplo de esta puede ser visto en la Figura 3.15.

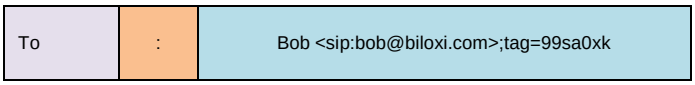


Figura 3.15: Cabecera To del Mensaje SIP

Via

Por último pero no menos importante tenemos la cabecera Via, esta se encarga de almacenar todo aquel servidor proxy que ha tenido injerencia con la solicitud. Por tal razón esta contiene el camino recorrido por la solicitud, esta información puede ser empleada en la detección de ciclos de enrutamiento. Otro uso aplicado de esta cabecera es el de enrutamiento de respuestas hacia el cliente quien ha generado una solicitud, para eso se hace uso del camino reverso generado por la inversión de las cabeceras acumuladas del tipo Via del mensaje SIP [1]. Un Ejemplo de esta puede ser visto en la Figura 3.16.

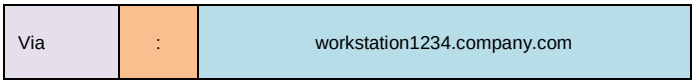


Figura 3.16: Cabecera Via del Mensaje SIP

3.4 Actores Inmersos en el Proceso de Comunicación

Ya hemos visto los elementos necesarios para el intercambio de información dentro del protocolo SIP, sin embargo estos solo conforman una porción no funcional del sistema estipulado dentro del RFC 3261 [1], es por ello que se debe realizar la definición de un conjunto de entidades adicionales, así como definir un conjunto de procesos que son llevados a cabo en el desarrollo del proceso de comunicación, esto para poder dar una visión más completa del panorama planteado por el RFC 3261 [1].

3.4.1 User Agent

Calificada como una entidad lógica que actúa como ambos usuario y servidor [7].

3.4.2 User Agent Client (UAC)

Conocido también como interfaz de interacción con el usuario final, también puede ser visto como una entidad lógica que crea o realiza una nueva solicitud, recordando siempre que el protocolo SIP está definido como un protocolo de capa de aplicación por lo que requiere de cierto nivel de interacción con el usuario para la realización de algunas tareas, bien sea de manera explícita o implícita [7]. Un caso sencillo que puede ilustrar este escenario es el expuesto en la Figura 3.17, donde se puede ver a un usuario (Bob) quien desea iniciar el proceso de comunicación, por lo que haciendo uso de la interfaz de interacción gráfica de su equipo, inicia aquel programa que reconoce como utilitario para establecer comunicación a través del protocolo SIP, en este caso un softphone o teléfono virtual manejado por tecnología VoIP, como lo es para el ejemplo Zoiper.

En la Figura 3.17 se puede apreciar como un usuario desea establecer una conexión con un ente par, por lo que ambos elementos localizados en los end-points del esquema, aquellos de ahora en adelante conocidos como UAC (User Agent Client), realizan los procesos de comunicación.

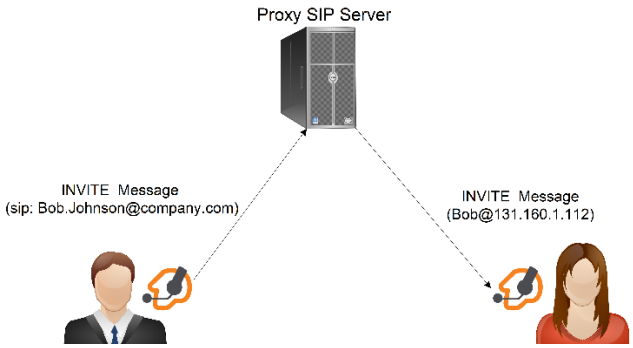


Figura 3.17: Envío de Solicitud INVITE por un Agente de Usuario

3.4.3 User Agent Server (UAS)

Dentro del conjunto de las principales entidades tipificadas como UAS se pueden identificar las siguientes:

- Servidores de Redirección (Redirect Server).
- Servidores de Proxy (Proxy Server).
- Servidores Registrar (Registrar Server).
- Servidores de Localización (Location Server).

Servidores de Redirección

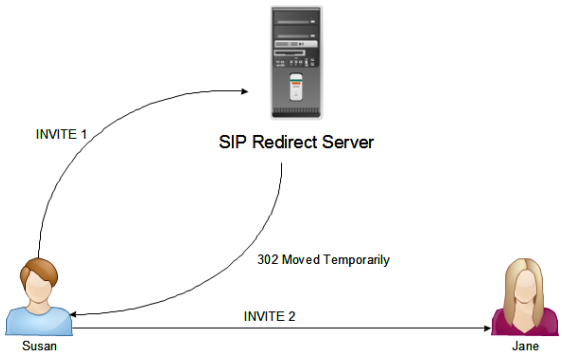


Figura 3.18: Escenario de Redirección

Los servidores de redirección ayudan con el proceso de localización de usuarios, proveyendo ubicaciones donde el usuario puede ser localizado. Para la realización de estas tareas los servidores de redirección hacen uso de los eventos del tipo 3xx como respuesta a solicitudes de Agentes de Usuario que resulten de interés, la respuesta generada por estos consiste en un URI dirigido al cliente que inicia el proceso de comunicación.

En la Figura 3.18, se representa el proceso de inicialización de comunicación entre dos entes pares UA, en la misma el proceso de comunicación es iniciado realizando el envio de solicitudes INVITE al servidor de redirección, a su vez (el ente Redirect Server) hace uso del servicio de localización para genera respuestas apropiativas sobre los entes solicitados, mediante el envío de mansajes con el código de estatus 302 Moved Temporarily sobre esta solicitud [7].

Servidores Proxy

En algunos casos las organizaciones pueden manejar determinadas políticas de seguridad en las cuales el flujo de datos entrantes y salientes es dirigido a servidores intermedios, que se encargan de aplicar ciertos filtros sobre la data para determinar su valides y permitir su enrutamiento. Estos servidores intermedios son denominados Servidores Proxy, cuya función es actuar como un agente intermedio mediante la adquisición del rol de cliente y servidor para permitir el reenvió de solicitudes y generación de respuestas.

En la Figura 3.19, el servidor proxy de la topología es conocido como un Gateway de acceso para ambos dominios, posiblemente ambos pertenecen al mismo conglomerado. Al momento de realizarse una solicitud de inicio de sesión por parte de un dispositivo en el dominio company.com la misma es enviada al proxy del dominio, que en esta representación cumple roles de Gateway y realiza tareas de Location Server para mantener el direccionamiento en el dominio enterprise.com.

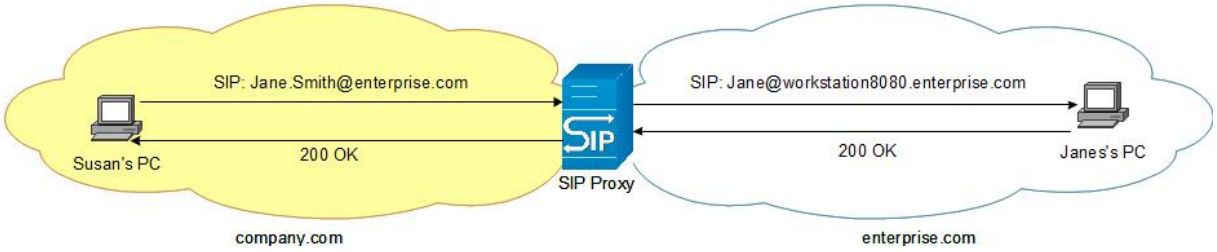
Como resultado envía un mensaje con el nombre de dominio en el campo URI de la línea de solicitud del mensaje SIP permitiendo que el mismo llegue al destinatario del mensaje. Posteriormente y gracias a que el cliente, Jane, atiende a la solicitud, se establece la conexión en el primer intento.

Servidores Registrar

También conocidos como servidores re registro, son los encargados de recibir y aceptar solicitudes del tipo REGISTER. Toda otra solicitud enviada a un servidor de registro recibe una respuesta del tipo 5xx Not Implemented, al menos siendo esto en servidores estándar basados en el RFC 3261 [1]. Luego de la recepción de estas

Figura 3.19: Representación del Funcionamiento de un Proxy SIP

solicitudes se hace visible a otros servidores dentro del mismo dominio administrativo, entre los cuales tenemos servidores proxy y de redirección [7]. En una solicitud de registro la cabecera To del mensaje SIP contiene el nombre del recurso a ser registrado dentro del servidor de registro y la cabecera Contact contiene el URI respectivo a ser asociado. Este evento crea lo que se conoce como un enlace temporal



o “binding” entre la dirección del registro o AOR (Address Of Record), el URI, y el valor de la cabecera Contact [7]. En la práctica es muy común que un UA deba autenticarse antes de la realización del enlace, esto previene que las llamadas entrantes no sean víctimas de ataques de Hijacking o secuestro de identidad por parte de un tercero que desee tomar la identidad de un recurso apuntándolo a un destino de su escogencia.

Servidores de Localización

En el entorno de las redes de computadores es común encontrarnos con infraestructuras en las que se realiza un control centralizado y global de datos de interés para un conjunto de entidades. Este comportamiento se mantiene en las redes SIP, en las cuales una organización, dada la existencia de un gran volumen de usuarios, decida colocar múltiples servidores SIP. Sin embargo esta infraestructura tiene un pormenor y es la complejidad inherente a la sincronización de datos a nivel

global para permitir que múltiples entes de servicio puedan conocer información heredada de sus pares. Una solución práctica consiste en la colocación de un servidor central donde los datos de localización son cargados y consultados, también conocidos como Servidores de Localización. De modo que servidores Registrar y servidores de Redirección realizan consultas y carga de datos a un solo repositorio centralizado, dando esto una visión general a todos los entes que tengas acceso a este servicio y mejorando el desempeño de la infraestructura [7].

4. Mecanismos de Alta Disponibilidad

En entornos reales, el establecimiento de múltiples sesiones en un mismo instante, dependiendo del servicio que se está ofreciendo u otros protocolos involucrados para mensajes multimedia como RTP (Real-time Transport Protocol), puede generar mucha carga y múltiples conexiones que pueden exceder las capacidades de los servidores que proveen el servicio. Por esto y posibles fallas a nivel de hardware o software que puedan generar problemas al servicio ofrecido, se utilizan mecanismos de alta disponibilidad para mantener el servicio proporcionado disponible la mayor cantidad de tiempo posible.

Para lograr el mantenimiento del servicio hay que tener ciertas definiciones claras, por ejemplo, conceptos básicos, fórmulas de medición y mecanismos de soluciones de alta disponibilidad. Estos conceptos serán desarrollados a continuación siguiendo principalmente las orientaciones e ideas de los autores P. Weygant [8] y E. Marcus [9].

4.1 Definición

El término Disponibilidad "…desde el punto de vista computacional, alude al periodo de tiempo durante el cual un servicio se encuentra disponible…". En cuanto a la expresión disponible, puede tratarse, tanto de tiempos de respuesta del servicio como de si se encuentra accesible para ser consumido por el usuario final [8].

Teniendo esto claro, entonces alta disponibilidad (High Availability) se refiere a "…un sistema que se encuentra continuamente operacional y disponible para el uso de los servicios provistos para los usuarios finales" [10]. Proveer alta disponibilidad de un sistema o servicio no es una tarea fácil, debido a que no cualquier servicio maneja mecanismos de alta disponibilidad. Por ello, ofrecer alta disponibilidad en un sistema requiere un diseño e implementación aparte, mitigando todos los puntos de falla que pueda tener un sistema. A lo largo de este capítulo nos referiremos a los distintos mecanismos para proveer alta disponibilidad y al modo de medir la disponibilidad de un sistema.

4.2 Conceptos Básicos

Antes de tocar las medidas y mecanismos de alta disponibilidad es importante tener en cuenta: qué significa y a qué se refiere el término tolerancia a falla y la expresión los 5 nueves.

4.2.1 Tolerancia a Falla

El término tolerancia a falla (Fault Tolerance) se entiende como "…un método para lograr altos niveles de disponibilidad". Se puede destacar que uno de los mecanismos y diseños mediante el cual podría darle a un sistema carácter tolerante a fallas es con redundancia. La redundancia es posible mitigando los únicos y cruciales puntos de falla de un sistema [8].

Por ejemplo, un servidor que esté ofreciendo un servicio web. Este servicio debe estar disponible el mayor tiempo posible con muy pocas caídas de servicio. Si este servidor solo tiene un disco duro instalado en él, significa que si falla ese disco se pierde toda la información del sistema y requerirá instalar un disco nuevo y reconstruir el servicio nuevamente. El hecho de tener un respaldo de la información podría disminuir el caos que puede generar una falla como esta; sin embargo, no es una solución óptima debido a que la misma es reactiva y, en conclusión, los tiempos de reparación del servicio serán muy altos. Por ello, lo más óptimo sería tener una redundancia a nivel de hardware, instalando así un segundo disco duro en el servidor y aplicando RAID 1 (Redundant Array of Independent Disks) entre ese arreglo de discos. Esto significa que ambos discos replicarán la información y ambos tendrán la información actualizada, y en caso de que un disco falle el otro se encargará de las funciones del primero.

El ejemplo aludido no significa que ya tenemos un sistema con todos los puntos de falla mitigados. Este simple ejemplo ofrece un único servicio, pero hay múltiples puntos de fallas que habría que tomar en cuenta, pues, además de redundancia, se pueden incluir otros mecanismos para aumentar la disponibilidad del servicio. Más adelante se hablará de los distintos mecanismos que se deben tomar en cuenta para diseñar un sistema con alta disponibilidad.

4.2.2 Los 5 Nueves

Cuando se habla del caso de "los 5 nueves" se hace referencia a una disponibilidad del 99,999% que para la mayoría de los servicios es ideal [8]. Sin embargo, dependiendo del tipo de servicio, este porcentaje podría ser menor y ser todavía óptimo.

Además de esto, las redes PSTN que son redes de hace muchos años y que se empiezan a considerar obsoletas por sus múltiples debilidades, tienen, no obstante, una gran fortaleza, ya que ofrecen alta disponibilidad y muy buena calidad. En las redes PSTN la disponibilidad, en promedio, es de 99,999% e incluso más. Eso significa 5 minutos de falla del servicio no planeada en todo un año.

Esta es una de las razones por la cual la telefonía IP no termina de solidificarse completamente en el mercado. La Tabla 4.1[1] muestra la cantidad de tiempo que un servicio se encuentra caído en distintos rangos y con diferentes porcentajes de

[1] https://en.wikipedia.org/wiki/High_availability

disponibilidad.

Availability %	Downtime per year	Downtime per month	Downtime per week	Downtime per day
90% (“one nine”)	**36.5 days**	**72 hours**	**16.8 hours**	**2.4 hours**
95%	18.25 days	36 hours	8.4 hours	1.2 hours
97%	10.96 days	21.6 hours	5.04 hours	43.2 minutes
98%	7.30 days	14.4 hours	3.36 hours	28.8 minutes
99% (“two nines”)	**3.65 days**	**7.20 hours**	**1.68 hours**	**14.4 minutes**
99.5%	1.83 days	3.60 hours	50.4 minutes	7.2 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes	2.88 minutes
99.9 %(“three nines”)	**8.76 hours**	**43.8 minutes**	**10.1 minutes**	**1.44 minutes**
99.95%	4.38 hours	21.56 minutes	5.04 minutes	43.2 seconds
99.99% (“four nines”)	**52.26 minutes**	**4.38 minutes**	**1.01 minutes**	**8.66 seconds**
99.995%	26.28 minutes	2.16 minutes	30.24 seconds	4.32 seconds
99.999% (“five nines”)	**5.26 minutes**	**25.9 seconds**	**6.05 seconds**	**864.3 milliseconds**
99.9999% (“six nines”)	**31.5 seconds**	**2.59 seconds**	**604.8 milliseconds**	**86.4 milliseconds**
99.99999% (“seven nines”)	**3.15 seconds**	**262.97 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% (“eight nines”)	**315.569 milliseconds**	**26.297 milliseconds**	**6.048 milliseconds**	**0.864 milliseconds**
99.9999999% (“nine nines”)	**31.5569 milliseconds**	**2.6297 milliseconds**	**0.6048 milliseconds**	**0.0864 milliseconds**

Tabla 4.1: Porcentaje de Disponibilidad Anual

4.3 Pérdida Inesperada de Servicio y Recuperación

Un Sistema inevitablemente tendrá pérdida del servicio, debido a que una pérdida de servicio puede ser tanto planeada como no planeada. Una caída de servicio planeada acarrea que el servicio se pare por diferentes razones, lo cual obliga a hacerle algún tipo de mantenimiento al sistema, para actualizarlo, mejorarlo o introducir algún cambio que requiera. En cambio, una caída de servicio no planeada es el tipo de caídas que se quieren evitar. Veremos que hay planes para mitigar estos errores y no permitir su prolongación al momento de repararlo.

4.3.1 Caídas de Servicio no Planeadas

Aquellas que pueden suceder en algunos escenarios similares a los siguientes:

- Falla de hardware.
- Error en el sistema de archivo.
- Error de kernel.
- Falla de disco.
- Picos de luz.
- Falla de poder.
- Falla de software o aplicaciones.
- Desastres naturales.
- Errores de administración o configuración.

Aunque por el tema abarcado en este trabajo importan los tipos de caída de servicio que están relacionados a componentes de hardware, de diseño o de software del

sistema, es relevante saber que no es siquiera la razón más recurrente a suceder. Las causas más comunes se deben a errores en configuración de red que puedan prohibir el uso del servicio, o algún tipo de error cometido por administradores u operadores que provoque la caída del servicio [8]. La Figura 4.1 sacada de [8] muestra una encuesta realizada por GartnetGroup en 1998 que mide las causas de caídas de servicio.

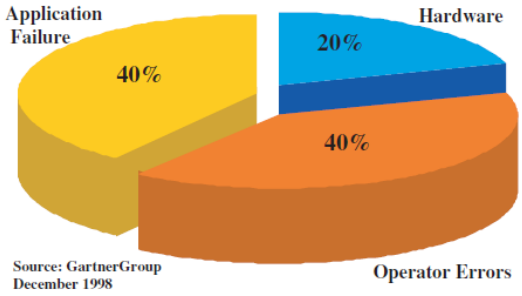


Figura 4.1: Porcentaje de Causas de Caídas de Servicio

4.3.2 Enfoque Proactivo al Fallo

Por lo anteriormente explicado ya se sabe que en un diseño de un sistema de alta disponibilidad es importante mitigar las caídas de servicio no planeadas, debido al desconocimiento de la razón de la falla. Para lograrlo es importante seguir los siguientes pasos:

4.3.2.1 Identificar los Puntos de Falla

“Disponibilidad puede ser visto como una cadena de servicios que debe permanecer intacta, las fallas son roturas en la cadena. Estos enlaces débiles son conocidos como puntos de falla” [8], la manera más efectiva de endurecer estos enlaces es creando redundancia o enlaces alternos de esos puntos, eliminando así los puntos de falla.

4.4 Configuración de Nodos

Habiéndose referido a estos conceptos básicos de disponibilidad es importante saber en qué topologías de alta disponibilidad se emplean estos conceptos. A lo largo de esta sección se verá la importancia del concepto de redundancia, que como ya fue explicado en algunos ejemplos es muy útil, aplicando distintos niveles de RAID a discos y redundancia en interfaces de red y fuentes de poder. Ahora, asegurar alta disponibilidad de un servicio no es tarea sencilla y con solo aplicar RAID o redundancia de interfaces no se asegura.

Por todo esto se ponen en práctica técnicas de failover, pero no solo a los niveles de los ejemplos anteriores, sino a nivel de nodos. Estos son dispositivos que forman parte de la solución que provee el servicio, comúnmente los servidores que se encuentran gestionando el servicio. Existen distintas técnicas de failover de estos nodos, técnicas para 2 nodos o para incluso granjas de nodos, como se explicará a continuación.

4.4.1 Modos de Operación de un Cluster de Dos Nodos

En la gestión de un cluster de dos nodos se pueden emplear dos tipos de configuraciones, activo-pasivo o activo-activo.

Activo-Pasivo

En un esquema activo-pasivo, uno de los nodos se encuentra procesando lo empleado por el servicio crítico que se desea exponer (considerándolo así el nodo activo), mientras que el otro nodo está en espera de manera dedicada en caso de que el nodo activo falle [9].

Activo-Activo

En un esquema activo-activo, ambos nodos se encuentran prestando servicios críticos distintos y, en caso de falla de alguno de los dos nodos, el restante deberá encargarse de la prestación de ambos servicios [9].

En la Figura 4.2 tomada de [9] se ve un caso de estudio en el cual dos servidores de aplicaciones, uno activo y otro pasivo, están interconectados por dos redes de **Heartbeat**, la cual es una herramienta que permite la creación y gestión de clusters, siendo también un protocolo de comunicación que permite captar la presencia de cada servidor incluido en el cluster. En esta arquitectura, Heartbeat desarrolla tareas de configuración del cluster, permitiendo determinar cuál nodo es el activo y cuál es el pasivo y, mediante un protocolo de comunicación, identificar si un servidor está habilitado o no. Además, configura lo que es llamado “IP flotante”, para configurar una IP virtual al cluster completo que se usa para que el usuario pueda acceder al servicio.

Adicionalmente, ambos servidores están conectados a dos controladores, gestionando los discos de almacenamiento para los servidores, los cuales se encuentran replicando la información entre ellos mediante un mecanismo de espejo. Ello permite, en tiempo real, replicar la información de cada controlador; así, en caso de que alguno de los controladores se encuentre inaccesible, los servidores accederán al otro controlador que mantiene la misma información que el controlador que se encuentra inaccesible [9].

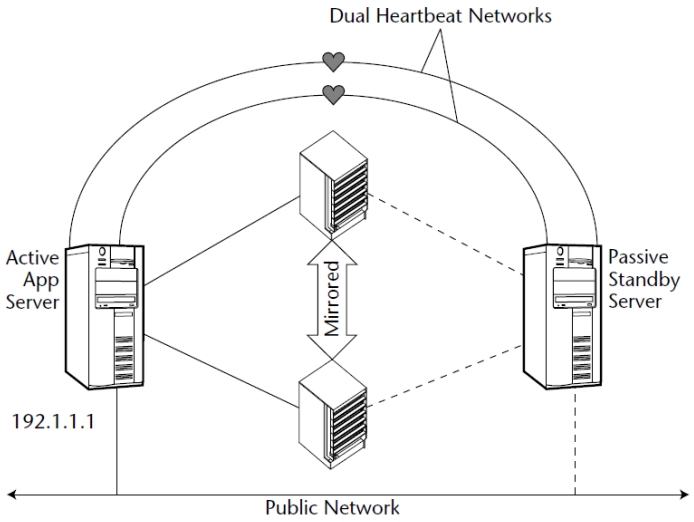


Figura 4.2: Topología de una Solución de Failover Activo-Pasivo

Los siguientes modos de operación hacen referencia a condiciones de contingencia en la cual se ha generado una eventualidad que impide el correcto funcionamiento de alguno de los servidores principales que prestan algún servicio.

4.4.1.1 Activo-Pasivo Failover

Cuando se genera una condición de contingencia el servidor pasivo que se encuentra en estado de no procesamiento, asume el control de los servicios provistos, estos son localizados y reiniciados en este servidor.

4.4.1.2 Activo - Activo Failover

Cuando una falla ocurre, el nodo secundario toma control de la prestación del servicio provisto por el nodo principal, este permanece como nodo activo hasta que el servicio es restablecido en el otro servidor.

Como la Figura 4.3 tomada de [9] refleja, la topología que se usa en soluciones de failover activo-activo y activo-pasivo es muy parecida. La diferencia radica en que, como ya fue explicado antes, ambos servidores se encuentran activos proporcionando un servicio crítico [9]. La Figura 4.3 refleja que para cada servicio proporcionado hay dos controladores aplicando técnicas de espejo para alta disponibilidad. Cada servidor tiene conexiones para tener acceso a ambos almacenamientos. Sí un solo servidor se encuentra activo, el mismo se encontrará proporcionando ambos servicios, por ello tendrá acceso a ambos almacenamientos y, dependiendo del servicio, accederá a uno de los almacenamientos o al otro. Este último planteamiento se puede ver claramente en la Figura 4.4 tomada de [9]:

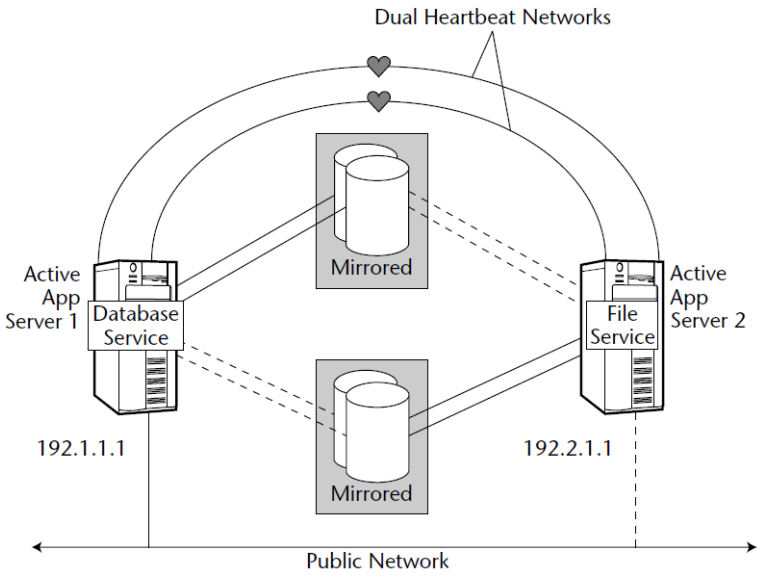


Figura 4.3: Topología de una Solución de Failover Activo-Activo antes de Aplicar Failover

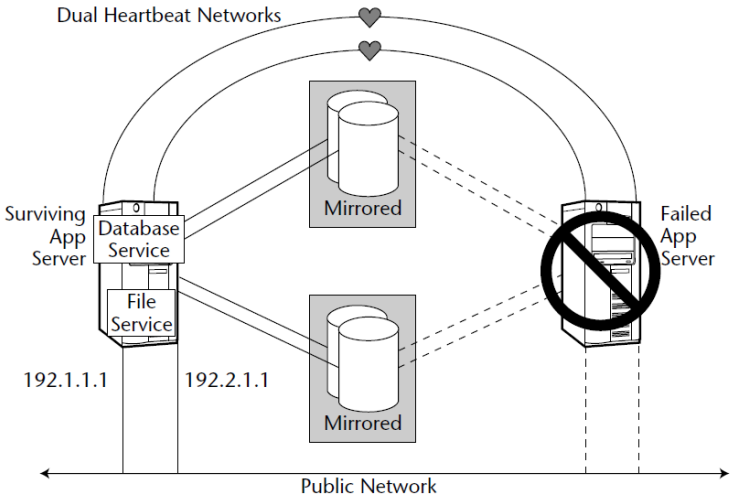


Figura 4.4: Topología de una Solución de Failover Activo-Activo después de Aplicar Failover

Para esta solución de failover se utilizan dos direcciones IP de acceso al público, uno para cada servicio. En el momento en que se caiga un servidor, el servidor activo deberá manejar ambas direcciones.

Soluciones de Failover Activo-Activo no se limitan en la utilización de múltiples servicios o que un mismo servicio sea provisto por ambos nodos. Ambos servidores pueden encontrarse prestando el mismo servicio, la diferencia radica en su diseño, como se trata a nivel de almacenamiento y de que utiliza técnicas de balanceo de carga como se verá más adelante.

Vista Comparativa para los Modos de Operación

- **Costos**: El mayor problema cuando se examinan los inconvenientes de failover activo-pasivo vs failover activo-activo son los costos fundamentalmente. Con activo-pasivo se deben aprovisionar dos servidores para hacer el trabajo de uno.

- **Configuración**: Las configuraciones activo-pasivo usualmente son las que ofrecen mayor disponibilidad con el tiempo sobre configuraciones activo-activo [9]. Esto se debe a que el servidor que se encuentra en estado pasivo no está haciendo algún procesamiento extra que pueda generar algún problema en este servidor.
- **Calidad de Servicio**: En este aspecto el hecho de tener un cuerpo de servidores donde existan múltiples instancias realizando tareas de procesamiento da un potencial superior, de acuerdo con los principios de la división de la carga de trabajo, Divide and Conquer, el modo Activo-Activo posee una clara ventaja.

5. Balanceo de Carga

Si se tienen dos servidores o nodos funcionando como centrales telefónicas con las mismas prestaciones, uno activo y otro pasivo, al momento en que el activo sufra alguna caída de servicio, el pasivo se encargará de ofrecer el mismo servicio. Ahora, si el activo se llegara a caer por un flujo de datos muy alto (múltiples llamadas gestionadas al mismo tiempo, inicio de llamadas, registro de usuarios, entre otros) que provoque un mal funcionamiento al servidor activo, es muy probable que le pasará los mismo al otro servidor en caso de ocurrir el failover y volverse activo, ya que son servidores con las mismas prestaciones. Para solventar esta clase de problemas es recomendable que, entre múltiples servidores, se distribuyan la carga. Para lograr esto se emplea el concepto de “Balanceo de Carga” (LB, Load Balancing).

5.1 Definición

El concepto de balanceo de carga consiste en que múltiples servidores se encarguen de ofrecer el mismo servicio y hagan el mismo trabajo. Para arribar a ello, usualmente se tiene un dispositivo físico con un software o un software encargado de la gestión de balanceo de carga. Este recibe todo el tráfico que va dirigido a los servidores que prestan el servicio y este mismo se encarga de gestionar, mediante diferentes técnicas y mecanismos, el redireccionamiento del tráfico al servidor que considere correcto.

5.2 Gestión de Granja de Servidores y Flujo de Básico

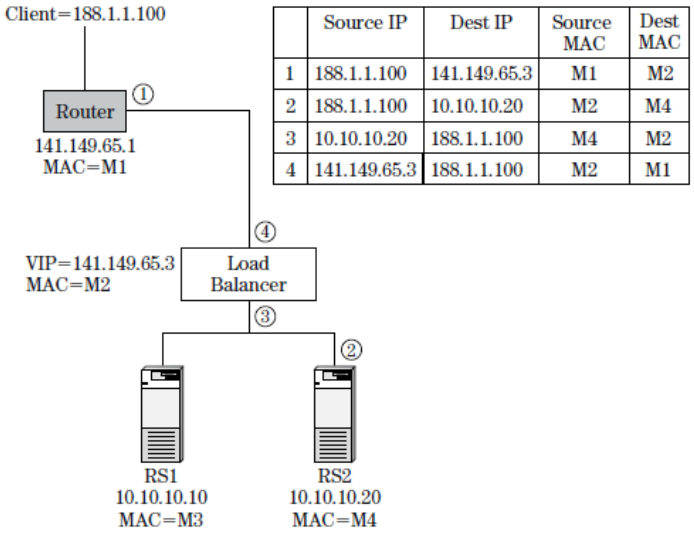
En la arquitectura de un modelo de balanceo de carga, se encuentran distintos servidores, definidos por el balanceador de carga como granja de servidores. Esta granja de servidores está directamente conectada al balanceador de carga o a un switch [11]. El balanceador de carga es el receptor principal de las peticiones realizadas por los distintos usuarios que desean consumir los servicios provistos por los servidores que se encuentran siendo balanceados.

El balanceador gestiona esta granja de servidores mediante una única dirección IP y simulando que se trata de un sólo servidor. Esta dirección IP es comúnmente llamada IP virtual y representa a toda la granja de servidores [11].

Una vez definidos la granja de servidores y los servicios a proveer, que formaran parte de la arquitectura de balanceador de carga, es necesario definir, por parte del balanceador de carga, cómo encaminar el flujo de comunicación de un servicio en particular al servidor que lo presta y, en caso de que el servicio sea brindado por múltiples servidores, dicho balanceador deberá proveer mecanismos para distribuir “equitativamente” la carga entre estos servidores, los cuales serán explicados en la Sección 5.4.

Ya precisadas, como queda expuesto, la IP virtual, la gestión de la granja de servidores

por parte del balanceador de carga y otros elementos (sobre los cuales se ampliará más adelante), se considera la arquitectura lista. Ahora, la Figura 5.1 tomada de [11] explica el flujo básico cuando la comunicación comienza por el cliente consumiendo un servicio gestionado por la arquitectura.



	Source IP	Dest IP	Source MAC	Dest MAC
1	188.1.1.100	141.149.65.3	M1	M2
2	188.1.1.100	10.10.10.20	M2	M4
3	10.10.10.20	188.1.1.100	M4	M2
4	141.149.65.3	188.1.1.100	M2	M1

Figura 5.1: Flujo Básico en Arquitecturas de Balanceo de Carga

Si se evalúa que los dispositivos de la granja de servidores están proveyendo un servicio Web mediante TCP, entonces en la tabla de sesión del balanceador de carga se agregan 4 valores que, a su vez, definen cualquier sesión de TCP. Esto permitirá almacenar la sesión TCP, para que el balanceador de carga mantenga la comunicación de esa sesión con el servidor elegido al momento que la sesión se genere exitosamente (como un balanceador de carga con estado).

- Dirección IP fuente.
- Puerto fuente.
- Dirección IP de destino.
- Puerto de destino.

Ahora, en el item (1) de la Figura 5.1, el flujo comienza con una petición HTTP por parte del cliente con dirección a la IP virtual (141.149.65.3) especificada por el balanceador. (2) En el momento que la petición llega al balanceador, éste identifica que se trata de una petición HTTP y una nueva sesión. Seguidamente, escoge el servidor RS2 (esta decisión es tomada por el mecanismo de distribución de carga utilizado) para enviárselo con dirección destino modificada a la IP privada de RS2, (este proceso es conocido como destination NAT o D-NAT). (3) Luego, RS2 envía la respuesta al balanceador de carga, pero con dirección destino al cliente. (4) Finalmente, el balanceador responde al cliente pero con la dirección fuente de la IP virtual (proceso llamado un-NAT).

5.3 Balanceador de Carga Sin Estado y Con Estado

Existen dos tipos de balanceo de carga, sin estado y con estado, la diferencia de éstos radica en que el balanceador de carga sin estado no mantiene ningún estado de sesión del tráfico distribuido en la arquitectura. En cambio, el balanceador de carga con estado mantiene el estado de cada sesión, así permitiendo al balanceador tomar decisiones por cada sesión.

5.3.1 Balanceador de Carga Sin Estado

Los balanceadores de carga sin estados suelen utilizar algoritmos de hashing. Estos algoritmos consisten en transformar ciertos datos tomados de las peticiones o paquetes de entrada en hash de valores muy bajos para determinar cuál servidor seleccionar de la granja [11]. Ello permite cierto nivel de persistencia; por ejemplo, si se utilizan mecanismos de hash basándose en la dirección IP fuente del cliente, todas las peticiones realizadas por ese cliente serán enviadas a dicho servidor. Existen múltiples métodos de hashing: Hash en base a IP fuente, Hash Buckets, etc.

5.3.2 Balanceador de Carga Con Estado

Los balanceadores de carga sin estado, utilizando mecanismos de hash, tienen ciertas limitaciones en la distribución equitativa de la carga. En tanto los balanceadores de carga con estado, como fue dicho anteriormente, mantienen el estado de las sesiones. Para lograr esto, el balanceador debe estar en la capacidad de determinar cuándo una sesión inicia y cuando termina. Cuando una sesión es iniciada, se determina el servidor de destino mediante mecanismos de distribución de carga. Una vez determinado, todos los siguientes paquetes que permanezcan a la misma sesión serán enviados al mismo servidor de destino hasta la culminación de la sesión [11].

Debido a que los estados de las sesiones son guardados en una tabla al momento que inician y borradas al momento que culminan, puede ocurrir el riesgo de que algunas sesiones queden inactivas y guardadas en la tabla por algún error en la comunicación de ambos entes. Por ello, los balanceadores de carga usualmente utilizan timers para eliminar sesiones que se encuentren inactivas.

5.4 Métodos de Distribución de Carga

Para la propuesta es importante evaluar cuál método de distribución de carga es el más apto para el proyecto.

5.4.1 Round-Robin

El método Round-Robin es uno de los métodos más populares y simples de distribución de carga. Su distribución se basa en otorgar cada petición a un servidor diferente hasta que todos hayan recibido; luego de que todos los servidores de destino

hayan recibido, el proceso se repite. Algunas de sus ventajas son su simpleza y el poco consumo de recursos del balanceador para realizar el algoritmo y su rápida respuesta en arquitecturas muy grandes con muchos dispositivos incluidos en la granja de servidores. Sin embargo, tiene la desventaja de proveer un mecanismo algo pobre de distribución de la carga, debido a que no posee mecanismos para evaluar la carga de cada elemento de la granja de servidores.

5.4.2 Elección por Carga

Esta técnica de balanceo de carga consiste en que, en la arquitectura, los servidores cuenten con la capacidad de poder proveer información de los indicadores (carga de CPU, de memoria, I/O de disco etc) que se deseen utilizar "…usando programas en cada servidor, conocidos como server-side agents, el balanceador de carga podrá detectar las condiciones de carga en el servidor a un nivel de detalle muy alto…" [11]. Todos estos indicadores medidos por estos agentes permiten al balanceador de carga determinar cuál servidor es el más adecuado para tomar la próxima petición.

Este tipo de técnicas exige que "el agente deba interoperar con la aplicación y cualquier otro componente que corra en el servidor". Además, varia, dependiendo del sistema operativo, como se accede a la información de los indicadores. Otras de sus desventajas es que genera una carga adicional a los servidores, debido a que tienen que extraer y estructurar esta información y además generan tráfico adicional a las redes [11].

Debido a estos inconvenientes, diferentes técnicas de balanceo de carga suelen ser combinadas. Por ejemplo, no solo se mide la carga de los servidores, sino también los que tengan menores conexiones o tiempos de respuesta menores y aplicar técnicas de peso[11].

Esta técnica es ampliamente usada en soluciones de aplicaciones web. Sin embargo, existen múltiples algoritmos y soluciones para lograr la elección de servidores de manera adaptativa y dinámica en soluciones multimedia y particularmente aquellos basados en SIP. Hay varias soluciones que aprovechan la naturaleza de SIP de mantener sesiones end-to-end y estar basado en transacciones, las cuales son iniciadas por la sesión mediante una petición INVITE y culminada con una transacción BYE. Por ello, múltiples algoritmos han sido tomados en cuenta para la elección de los servidores, ya que el balanceador de carga debe ser consciente de mantener la asignación de esa petición y guardar información de esa sesión (Session-aware Request Assignment, SARA) [12]. Ello, para que el usuario que haya iniciado sesión con algún servidor, pueda seguir comunicándose con ese servidor. Con esto, es posible saber cuántas sesiones activas están siendo procesadas por cada servidor. Algunos ejemplos de algoritmos para balancear transacciones SIP que, en este caso, son definidos por H. Jiang [12] son:

- **Call-Join-Shortest-Queue (CJSQ):** “Este algoritmo rastrea el número de sesiones asignado a cada servidor back-end y enruta nuevas llamadas SIP al nodo que tenga menos número de llamadas activas” [12].
- **Transaction-Join-Shortest-Queue (TSJQ):** Este algoritmo “…enruta nuevas llamadas al servidor que tiene menos transacciones activas” [12].

5.4.3 Parámetros de Carga de Servidores

Este método es utilizado para definir parámetros que son considerados como límites antes de percibir degradación en los servicios. En ciertos casos, determinados parámetros son más importantes que otros en diferentes escenarios. Algunos de los parámetros de carga más considerados son: porcentaje de procesamiento, porcentaje de memoria RAM, cantidad de conexiones, cantidad de transacciones o llamadas activas (en caso de soluciones SIP es muy utilizado). Éste método es usualmente utilizado en combinación con alguno de los otros métodos explicados.

Por ejemplo, se puede estipular un escenario donde dos servidores encargados de funciones de telefonía se encuentran siendo balanceados. Allí, el balanceador podría definir (siempre y cuando mantenga el estado de las sesiones o al menos el Call-ID de las llamadas) que si la cantidad de llamadas activas en uno de los servidores superan las 1000 llamadas; ese servidor no recibirá más carga por determinado tiempo establecido por el balanceador o hasta que se reduzca esa cantidad hasta cierto punto.

Cada método tiene tanto ventajas como desventajas, dependiendo muchas de la necesidad de la solución. Soluciones de balanceo de carga combinadas que permitan evaluar la carga de los nodos es, sin duda, de las que ofrecen mayor escalabilidad y eficiencia en tiempo real, evitando así fallas en el servicio, pero conlleva a complicaciones en cuanto a implementación y desarrollo y además de la carga extra que el balanceador experimenta por los algoritmos a realizar. Por ello, la evaluación de las necesidades que debe cumplir el servicio que se desea proveer, más la confiabilidad que ofrece la solución de infraestructura utilizada para proporcionar el servicio, es importante para tomar la técnica más conveniente para la solución.

5.5 Técnicas de Balanceo de Carga

Existen diferentes escenarios para definir cómo se comporta una arquitectura de balanceo de carga. A lo largo de esta sección se explican algunas de estas técnicas que definen el comportamiento de dicha arquitectura.

5.5.1 Traducción de Direcciones IP (NAT)

En esta técnica de balanceo de carga, se aprovecha NAT (Network Address Translation) para exponer una única dirección para el uso del servicio y el balanceador se encarga de traducir esa dirección en alguna de las direcciones privadas que cada servidor tiene configurada. Esta solución es implementada en caso de que en las técnicas de enrutamiento directo no sea posible su instrumentación o provoque alguna ineficiencia en la topología existente donde se desee desplegar.

Existen distintos tipos de mecanismos de NAT en ambientes de balanceo de carga, principalmente los siguientes:

Destination NAT (NAT de Destino)

La Figura 5.1 la cual expresa el flujo básico del consumo de un servicio en una arquitectura de balanceo de carga, utiliza este mecanismo. El mismo consiste, al momento que una petición de un usuario llega al balanceador, en que el balanceador modifica la dirección IP de destino de la petición (usualmente conocida como IP virtual) y coloca la dirección IP privada del servidor dentro de la granja escogido para proveer el servicio. Ahora, en el instante en que el servidor responde a esta petición al balanceador, el balanceador, éste deberá cambiar la dirección IP fuente por la dirección IP virtual del balanceador (aplicando un-NAT).

Source NAT (NAT Fuente)

El Source NAT es un proceso parecido al Destination NAT. La diferencia radica en que, además de requerir la traducción de la IP de destino al momento de una petición del usuario, también es requerida la traducción de la IP fuente por la dirección IP privada del balanceador; este proceso es también llamado full-NAT [11]. El mismo proceso ocurre en el instante en que el servidor de la granja responde: la dirección IP de destino de la respuesta del servidor contiene la dirección IP privada del balanceador, siendo la misma cambiada por la dirección IP de destino del usuario para transmitir la respuesta al usuario.

Enhanced NAT (NAT Perfeccionado)

“El termino enhanced NAT es utilizado para describir los mecanismos de NAT realizado por el balanceador de carga con conocimiento específico del protocolo para que ciertos protocolos funcionen con balanceo de carga” [11]. Este proceso es muy común en escenarios donde los servicios provistos son gestionados por protocolos de tiempo real, principalmente negociados, previamente, por algún protocolo de establecimiento de sesiones.

Este mecanismo es ampliamente utilizado en escenarios donde el protocolo SIP es el encargado para el establecimiento de sesiones, el proceso de resolución de estos problemas es mejor conocido como NAT Traversal. La manera en que estos inconvenientes son resueltos es mediante la capacidad del balanceador de modificar

ciertos parámetros a nivel de la cabecera SIP e incluso SDP (en caso de describirse los parámetros para el flujo de datos en tiempo real, como en el protocolo RTP).

La implementación de NAT en arquitecturas de balanceo de carga para el protocolo SIP tiene ciertos problemas. Esto es debido a que “el mismo NAT se vuelve el cuello de botella, volviendo la arquitectura ineficiente” [13]. Además, debido a la naturaleza de SIP de mantener el estado de las transacciones en los endpoints (B2BUA, Back-to-Back User Agent), porque las retransmisiones deben ser manejadas por el mismo servidor SIP interno, el balanceador, aplicando NAT, deberá mantener una tabla de sesión [13].

La Figura 5.2 muestra una topología en la cual los servidores que se encuentran siendo balanceados, están en una red aislada y privada del resto de la topología de red.

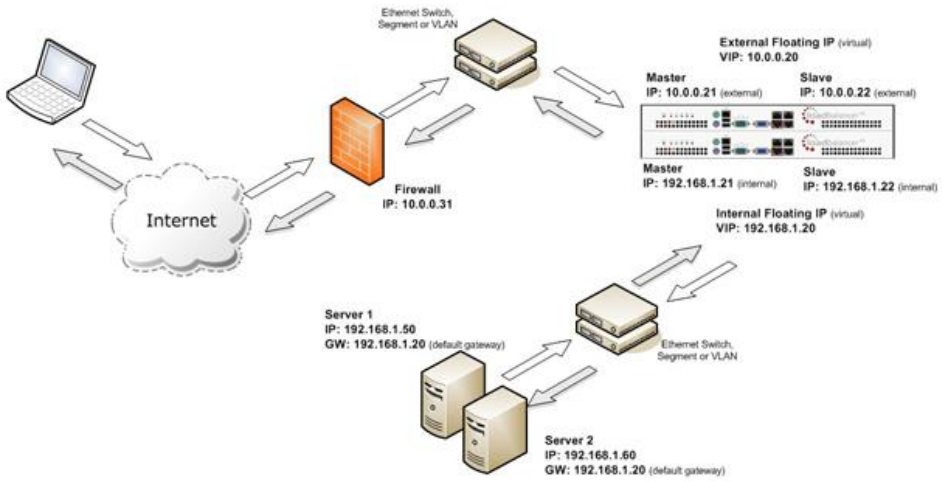


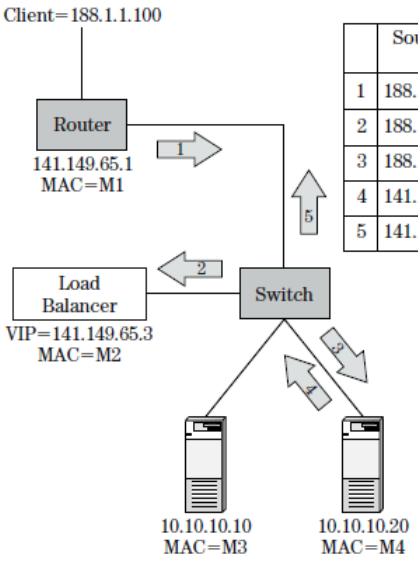
Figura 5.2: Topología de Implementación de NAT en un Esquema de Balanceo de Carga

5.5.2 Enrutamiento Directo

En esta técnica, el balanceador de carga y los servidores se encuentran en una misma subred y además los servidores redundantes usan la misma dirección IP. “Tiene la enorme ventaja de no modificar nada al nivel de IP, así los servidores pueden responder directamente al usuario sin necesidad de pasar por el balanceador, esto es llamado DSN (Direct Server Return)” [14]. Esta técnica es recomendada con el uso de proxies SIP sin estado que usan UDP como protocolo de capa de transporte y tratando cada petición como independiente, sin mantener estado de la transacción en el proxy.

Para lograr esto, tanto el balanceador como los servidores de la granja deben estar en el mismo dominio de capa 2 para basar su comunicación, simplemente traduciendo direcciones MAC de destino en caso de flujos de peticiones de los usuarios. Y en el momento de responder por parte del servidor, es necesario evitar el proceso de un-NAT, pero respondiendo directamente con la IP virtual como dirección IP fuente. Para resolverlo, se configura la IP virtual como dirección IP en la interfaz de loopback de

cada servidor de la granja. En la Figura 5.3 tomada de [11] se puede ver el flujo de los paquetes y las direcciones utilizadas.



	Source IP	Dest IP	Source MAC	Dest MAC
1	188.1.1.100	141.149.65.3	M1	M2
2	188.1.1.100	141.149.65.3	M1	M2
3	188.1.1.100	141.149.65.3	M2	M4
4	141.149.65.3	188.1.1.100	M4	M1
5	141.149.65.3	188.1.1.100	M4	M1

Figura 5.3: Flujo de Mensajes en Técnicas de Enrutamiento Directo

5.6 Health Checks

Ahora, ya se saben los distintos métodos de distribución y técnicas de balanceo de cargas que pueden ser utilizados, pero ¿qué pasa si ocurre alguna falla en alguno de los nodos que se encuentran siendo balanceados? Si el balanceador no está en la capacidad de identificar que está fallando algún nodo, éste seguirá reenviando peticiones a ese nodo produciendo así fallas en el servicio desde la perspectiva del usuario. En cambio, si el balanceador está en la capacidad de verificar el estado de los nodos que se encuentra balanceando, los nodos afectados no recibirán ninguna petición por parte del balanceador. Es por esto que hacer health checks es importante en un esquema de balanceo de carga.

5.6.1 Definición

Health Checks es un mecanismo utilizado en soluciones de balanceo de cargas para verificar, por parte del balanceador, la disponibilidad de los servidores que se encuentra gestionando. Esto evita la presencia del troubleshooting manual por parte de administradores, para levantar el servicio. Ello, debido a que el balanceador se enterará de que un servidor de su granja de servidores se encuentra inactivo y utilizará alguno de los activos para tomar la petición del usuario. Así, el administrador puede monitorear el servidor caído mientras el servicio se encuentra activo para el usuario.

5.6.2 Health Checks por Capas

Hay diferentes mecanismos para aplicar health checks. Casi todos ellos son, específicamente, por capas (OSI Layers). Otros utilizan mecanismos complejos de health checks a nivel de capa 7, los siguientes puntos explicarán el funcionamiento de cada una:

- A nivel de capa 2, la verificación se basa en peticiones ARP (Address Resolution Protocol) para encontrar la dirección MAC dando la dirección IP. Debido a que el balanceador es configurado con la información de las direcciones IP reales de los servidores, el balanceador envía mensajes ARP por cada dirección IP real de los servidores que gestiona. El servidor responderá a la petición ARP al menos que esté caído [11].
- A nivel de capa 3, la verificación se basa en hacer pings a las direcciones IP reales de los distintos servidores de la granja que el balanceador gestiona. En caso de que no responda el servidor con Echo Reply, se asume caído.
- Tanto a nivel de capa 2 como al nivel de las capas 3 y 4, el mecanismo es muy parecido; su diferencia está en lo que maneja cada capa. Capa 2 maneja direcciones físicas, por ende aprovecha el protocolo ARP para validar si responde. En capa 3 se manejan las direcciones IP directamente, así que aprovecha el protocolo ICMP (Internet Control Message Protocol) para mandar un Echo Request y valida recibiendo el Echo Reply correspondiente. En caso de capa 4 utiliza el protocolo TCP o UDP para la verificación; por ejemplo, enviando un TCP SYN a una dirección IP real de uno de los servidores con el puerto al cual escucha el servicio que se está utilizando; en caso de no recibir respuesta (TCP SYN + ACK) se asume caído el servidor.
- A nivel de capa de aplicación, es mucho más complicado y a su vez mucho más efectivo para verificar la disponibilidad del servidor. “No hay reglas sobre cuán extensivo los health checks, a nivel de aplicación, deberían ser, y varía entre los diferentes productos de balanceo de carga”. Debido a la cantidad de protocolos a nivel de aplicación que existen y las maneras en que pueden ser aplicados, esta verificación usualmente queda de parte de los fabricantes, quienes deberán optar por las técnicas que les parezcan más óptimas [11].

Si vamos al caso de estudio, que es el protocolo SIP, algunos fabricantes basan los health checks mediante procesos de petición – respuesta (ver Tabla 5.1 [15]). Dependiendo de la petición y las distintas respuestas posibles a esa petición (la cual los fabricadores suelen basarse en el código de la respuesta), el balanceador determina si ese servidor está activo o inactivo.

SIP Response Message Types	Description
1xx	Information Responses – For Example: 180, Ringing
2xx	Succesfull Responses – For Example: 200, OK
3xx	Redirection Responses – For Example: 302, Moved Temporarily
4xx	Request Failures Responses – For Example: 403, Forbidden
5xx	Server Failure Responses – For Example: 504, Gateway Time-out
6xx	Global Failure Responses – For Example: 600, Busy Everywhere

Tabla 5.1: Posibles Respuestas a una Petición REGISTRAR

Tomando como ejemplo una de las especificaciones de health check de unos switches especializados para ofrecer balanceo de carga por SIP de IBM, llamados BladeCenter, al recibir una petición REGISTRAR al switch, él reenvía esa petición REGISTRAR al servidor. “El switch busca, por el tipo de respuesta del servidor 1xx, 2xx, 3xx o 4xx, para determinar si el servidor está activo, si el switch recibe una respuesta de tipo 5xx o 6xx, el switch declara al servidor como caído” [15].

5.7 Persistencia

A lo largo del capítulo de balanceo de carga se han formulado consideraciones sobre diferentes métodos de distribución de carga y técnicas de balanceo, balanceo por DNS con registros SRV, enrutamiento directo, elección por carga adaptativa, etc. En algunas de éstas, se hizo referencia a la naturaleza de SIP de mantener estado de transacciones y manejo de sesiones por parte de SIP. Sin embargo no se ha hecho énfasis ni en las causas ni en el gestionamiento de éstas; por ello, a continuación se realiza un análisis explicativo relacionado con la persistencia de estas sesiones.

5.7.1 Persistencia de Sesiones

Se puede definir la persistencia de sesiones como “la habilidad de mantener todas las sesiones de ciertos usuarios al mismo servidor por la duración de la transacción de esa aplicación” [11]. Hay distintos mecanismos para esta persistencia de sesiones, los cuales usualmente dependen de los protocolos involucrados y de las características de ellos. Esto, debido a que un balanceador de carga debe saber 2 cosas para poder aplicar persistencia de sesiones: (1) como identificar a un usuario, y (2) como reconocer cuando una transacción de aplicación comienza y culmina [11].

En el caso de estudio de este trabajo, el cual se propone el uso del protocolo SIP, ya se sabe que SIP es un protocolo basado en sesiones y tiene definido, mediante transacciones INVITE y BYE, cuándo una sesión comienza y cuándo culmina, respectivamente. Debido a que usualmente es preferible usar el protocolo UDP, el mismo protocolo SIP ofrece mecanismos para identificar una sesión utilizando el campo CALL-ID o incluso definiendo persistencia de dialogs, como ya fue explicado en la Sección 3.10 de este trabajo.

Muchos balanceadores de carga ofrecen mecanismos de persistencia dependiendo del algoritmo de balanceo de carga que utilicen. Aquellos que utilizan mecanismos de hash como método de balanceo de carga, que consiste en algoritmos que aplican pesos sobre los distintos servidores de manera estática, con las mismas ventajas y desventajas del método descrito en la sección 5.3.1, usan el mismo resultado de hash para mantener la sesión. Otros algoritmos suelen mantener una tabla de sesiones. [12].

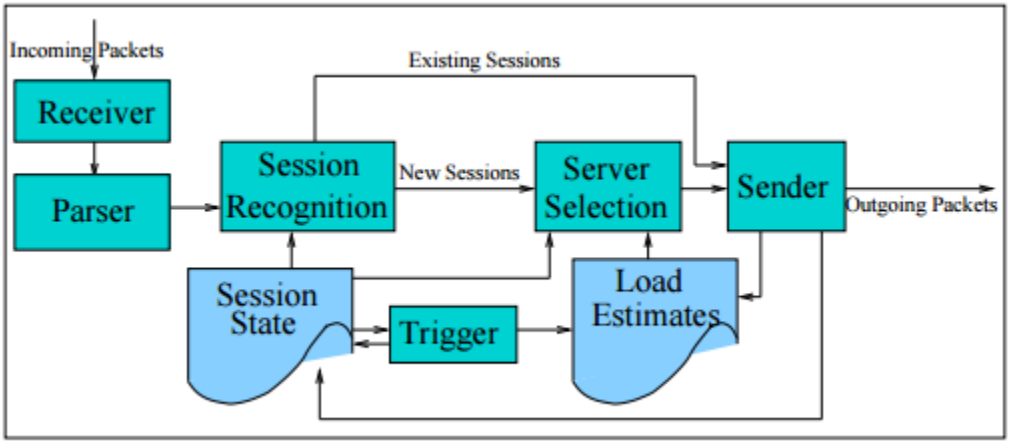


Figura 5.4: Flujo de Balanceo de Carga con Persistencia

Hasta aquí se ha comentado acerca del protocolo SIP, mecanismos de alta disponibilidad y esquemas de balanceo de carga, ahora queda comentar sobre herramientas disponibles para proveer una solución de alta disponibilidad más balanceo de carga de telefonía confiable. Aunque soluciones de alta disponibilidad y balanceo de carga confiables podrían considerarse más confiables con la utilización de dispositivos propietarios, soluciones de alta disponibilidad como la pila de clustering Corosync + Pacemaker + DRBD y soluciones de balanceo de carga básicas que ofrece el módulo de dispatcher de Kamailio como su capacidad de recepción de conexiones SIP y soluciones de telefonía como Asterisk permitirán desarrollar la solución de alta disponibilidad de telefonía IP software libre que se desea.

6. Herramientas de Interés

A lo largo de este trabajo, se ha hecho énfasis en los ciertos elementos, desde el punto de vista teórico, relacionado al campo de tecnología de VoIP, específicamente el establecimiento de comunicaciones. Además, se ha hecho énfasis en los mecanismos de alta disponibilidad y de contingencia que permitirían el desarrollo de una solución de telefonía escalable y con capacidad de ofrecer alta calidad en su servicio provisto. En este capítulo se hará énfasis en las herramientas que permitirán la realización de la solución de telefonía que se desea implementar.

6.1 Asterisk

Soluciones de telefonía consisten en múltiples elementos, como el establecimiento de comunicaciones y el flujo de comunicación del mismo. Si bien en este trabajo se ha hecho mención principalmente al protocolo que permite el establecimiento de una comunicación en ambientes de VoIP y el cual será principalmente gestionado por otra herramienta que se mencionará próximamente, la gestión del flujo multimedia debe ser también tomado en cuenta. Es en herramientas de este estilo donde el punto anteriormente mencionado es gestionado de manera eficiente y con múltiples funcionalidades como se verá a lo largo de esta Sección.

6.1.1 Definición

Asterisk se puede definir vagamente como una PBX de software libre. Alguna de las capacidades, basándose en lo explicado por F. Goncalves [16], que están incluidas en las instalaciones de Asterisk encima de soluciones de PBX comunes son:

- Interconectar diferentes oficinas entre distintas redes, así estén en Internet (redes externas diferentes).
- Implementar aplicaciones como IVR's (Interactive Voice Response) que se conecte con distintas aplicaciones.
- Dar acceso a la PBX de Asterisk a cualquier usuario que se encuentre fuera de la oficina mediante conexiones VPN (Virtual Private Network) por ejemplo.
- Creación y reproducción de música en espera para las colas de llamadas.
- Crear sistemas de Call Center sencillos (Asterisk te da la facilidad de poder modificarlos a cualquier necesidad debido a ser software libre).

6.1.2 Arquitectura

La arquitectura de Asterisk es muy diferente a otras PBX tradicionales. Por ejemplo, la estructura de Asterisk, a nivel lógico, no diferencia a los dispositivos de telefonía como faxes, teléfonos, etc, o líneas entrantes como troncales o líneas ISDN (Integrated Services Digital Network), al igual que lo hacen las PBX. Asterisk los ve como un componente que es gestionado como un elemento que pasa por los canales definidos

en Asterisk. Igualmente, los canales, aunque con sus diferencias, son gestionadas por el plan de discado (Dialplan) de manera muy similar, lo que permite que recursos externos o ubicados en el exterior puedan ser gestionados como recursos internos. Esto da una libertad y facilidad al momento de configurar los componentes [17]. La Figura 6.1 tomada de [17], expresa claramente lo explicado en este párrafo:

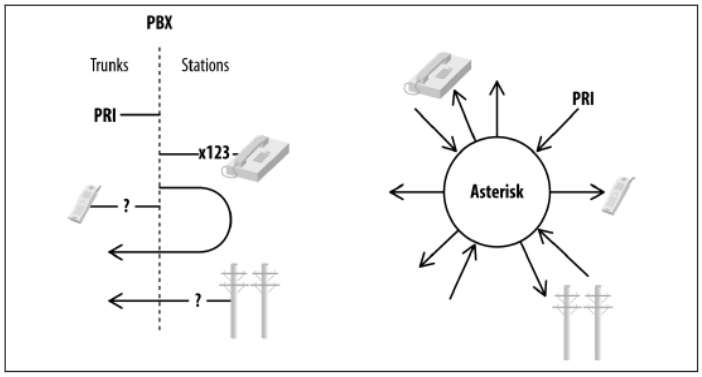


Figura 6.1: Diferencia de Abstracción entre PBX Convencionales y Asterisk

Desde el punto de vista general, la arquitectura de Asterisk se divide en tres grupos como muestra la Figura 6.2 tomada de [18]:

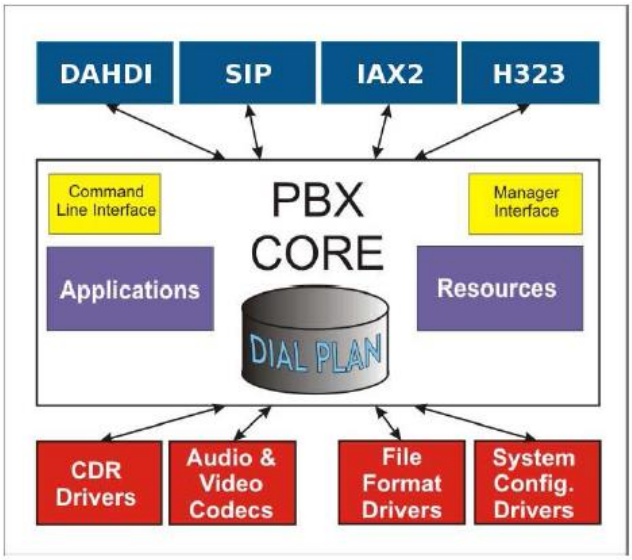


Figura 6.2: Arquitectura de Asterisk

Los componentes en color azul son los Drivers de Canal, los que se encuentran en color rojo son Drivers y Convertidores y los que se encuentran dentro del cuadro constituyen el Núcleo PBX.

Núcleo PBX

El núcleo PBX es el corazón de Asterisk y es el que gestiona la mayoría de las

funcionalidades del sistema. Entre algunas de sus funcionalidades están:

- Lectura de archivos de configuración, incluyendo Plan de Discado.
- Carga de módulos.
- Procesamiento de solicitudes del Plan de Discado.
- Gestión de las aplicaciones.
- Creación de instancias de canal.

Drivers de Canal

Los drivers de canal son aquellos que se encargan de gestionar las líneas de entrada y salida con el exterior. Asterisk utiliza el núcleo para procesar las llamadas por el plan de discado y utiliza los drivers de canal para gestionar el canal que utilizará y encargarse de la comunicación como tal. Más adelante se hace énfasis en la funcionalidad de éstos.

Drivers y Convertidores

Asterisk es un sistema con forma modular, tanto los drivers de canal como los drivers y convertidores son módulos. Éstos son llamados por el núcleo de Asterisk dependiendo de la necesidad. Hay drivers que se encargan de la validación de configuraciones del sistema, otros para generación de tipos de archivo como también para conversión de ellos utilizando el núcleo de Asterisk. Además se incluyen los convertidores, mejor conocidos como codecs.

Módulos

Los módulos proveen funcionalidades como “las de los drivers de canal (chan_sip.so) o recursos que permiten la conexión con tecnologías externas (func_odbc.so). Los módulos de Asterisk son cargados basados en lo que este especificado en el archivo /etc/asterisk/modules.conf” [17]. Existen distintos tipos de módulos, entre ellos están:

- Módulos de Aplicaciones.
- Módulos de Bridging.
- Módulos de Grabación de Llamadas (Call Detail Record, CDR).
- Módulos de Drivers de Canal.
- Módulos de Traductores de Codecs.
- Módulos de Interpretación de Formatos.
- Módulos de Funcionalidades del Plan de Discado.
- Módulos de PBX.

Aplicaciones

Las aplicaciones son funciones utilizadas para su ejecución en el plan de discado. La aplicación más conocida e importante es Dial() que se encarga de crear una llamada a un canal según los parámetros especificados [18]. Entre las características comunes definidas por [18] están:

- Las acciones de las aplicaciones están enfocadas por y para los canales.

- Se cargan de forma dinámica.
- Se ejecutan de manera síncrona.

Recursos

Los recursos son muy parecidos a las aplicaciones, pero estos son cargados de manera estática en vez de dinámica y "…pueden operar simultáneamente en múltiples canales, en vez de crearse dinámicamente para cada canal en curso" [18]. Un ejemplo de un recurso es aquel que permite realizar interconexiones con base de datos a través de ODBC (Open Database Conectivity). El formato de estos módulos es "res_<nombre>.so".

Canales

"Un canal es equivalente a una línea de teléfono, pero en formato digital" [16]. En los canales se utilizan protocolos de señalización (SIP, IAX2) y combinación de codecs (G.711, G.729) para permitir la comunicación de voz por medios digitales. Múltiples tipos de canales son soportados en Asterisk, como IAX2 (Inter-Asterisk eXchange Protocol) y H.323, pero en este trabajo se hace principal énfasis en SIP.

Codecs

Los codecs posibilitan la comunicación de voz entre entes por medios digitales y están en la capacidad de comprimir la voz, así aprovechando el ancho de banda de los medios, como también de optimizar la restauración de la voz en el destino.

Algunos codecs pueden consumir altas cantidades de CPU para aumentar la compresión o el muestreo de la voz. Es importante tener en cuenta cual es la conveniente para la plataforma que se va a utilizar. Algunos de los codecs soportados por Asterisk son:

- G.711 ulaw.
- G.711 alaw.
- GSM (Global System for Mobile).
- Speex.

Plan de Discado

El Plan de Discado es el componente más importante de Asterisk, ya que es el que se encarga de gestionar las acciones orientada a los canales, como, por ejemplo, el enrutamiento de llamadas En las centrales con Asterisk, la mayoría de la configuración relacionada con el Plan de Discado se realiza en un archivo predeterminado con el nombre de extensions.conf dentro del directorio /etc/asterisk.

Su función se basa en parámetros pasados por llamadas entrantes. Estos parámetros son pasados al realizar una llamada para verificar el archivo y, según la relación, ejecutar la función.

6.2 Kamailio

Kamailio, junto a herramientas para la administración de tráfico multimedia principalmente por voz como Asterisk y FreeSWITCH, permiten implementar soluciones altamente escalables de telefonía y además aliviar la carga a las referidas herramientas con servicios de registro y de gestión de conexiones directas por UAs filtrando cada paquete SIP.

6.2.1 Definición

Kamailio es un servidor SIP software libre fundado por el mismo equipo del anteriormente llamado OpenSER, de donde se toma la mayoría de la base del código para la formación de Kamailio y OpenSIPs. Kamailio y OpenSIPs son servidores SIP similares, sin embargo tienen filosofías diferentes.

Kamailio, debido a que posee una alta capacidad de gestionar miles de establecimientos de llamada por segundo y múltiples funcionalidades a nivel del protocolo SIP, como filtrado de paquetes SIP y distribución de llamadas, lo convierten en una herramienta ejemplar a utilizar en soluciones de comunicaciones en tiempo real basadas en SIP. Desde el punto de vista general, Kamailio ofrece funcionalidades de los siguientes tipos de servidores SIP, como planteado en el RFC 3261 [1]:

- Registrar Server.
- Location Server.
- Proxy Server.
- SIP Application Server.
- Redirect Server.

6.2.2 Arquitectura

De igual manera que Asterisk y FreeSWITCH, Kamailio consiste en un sistema de forma modular. El núcleo maneja únicamente funcionalidades básicas de un servidor SIP y Registrar, y el resto de las funcionalidades son añadidas mediante módulos incluidos en un archivo de configuración central de Kamailio. Algunas de las funcionalidades del núcleo de Kamailio son:

- Gestor de la capa de transporte.
- Gestión de memoria.
- Sistema de bloqueo.
- Intérprete de los archivos de configuración (desde el punto de vista sintáctico).

6.2.3 Módulos

Kamailio desde su creación, tiene alrededor de 150 módulos listos para ser incluidos con distintas funcionalidades que hacen a Kamailio una herramienta potente. Información de estos módulos son obtenidas de la página oficial de Kamailio donde sus

módulos y funcionalidades son los siguientes[2]:

- **APP_<Lenguaje>**: Estos módulos permiten la compilación, interpretación, ejecución, etc; de códigos desarrollados en dicho lenguaje y son accedidos mediante interfaces desarrolladas para utilizar el módulo. Algunos de los módulos son: APP_PYTHON, APP_JAVA y APP_PERL.
- **ACC**: Este módulo permite contabilizar información relacionada con múltiples transacciones y almacenarlas en base de datos o syslog por ejemplo. Está en la capacidad de mantener reportes relacionado a transacciones SIP tipo INVITE, estados de un Dialog e incluso de CDR's, así manteniendo reportes de cuando comenzó una llamada, cuando terminó y su duración.
- **Auth_identity**: Módulo que provee funcionalidades para la identificación segura de los originadores de mensajes SIP siguiendo lo especificado en el RFC 4474 [19].
- **DB_<Manejador de Base de Datos>**: Módulo para el soporte de múltiples manejadores de base de datos mediante la implementación de un API. Algunos de los módulos son: DB_MONGODB, DB_MYSQL, DB_POSTGRES, DB_ORACLE, entre otros.
- **DIALOG**: Módulo utilizado para proveer a implementaciones de proxy con estado con Kamailio la capacidad de lo mantener estados de dialog, en pocas palabras, de llamadas y no solamente de transacciones SIP como hace con el módulo TM.
- **DIALPLAN**: Módulo que permite la construcción de planes de discado.
- **DISPATCHER**: Este módulo es una pieza clave en la implementación de una solución escalable y de alta disponibilidad de calidad, debido a la capacidad de ofrecer balanceo de carga de mensajes SIP, no solo a nivel de capa 3 o 4 sino también a nivel de capa de aplicación. En pocas palabras, este módulo está en la capacidad de ofrecer balanceo por distribución de llamadas y además puede ofrecer funcionalidades de alta disponibilidad redistribuyendo a otro nodo en esquemas activo-pasivo si está en combinación del módulo TM.
- **NAT_TRAVERSAL**: Módulo que incluye funcionalidades para detectar UAs detrás de NAT modificando cabeceras SIP y utilizando mecanismos de keepalive contra el NAT para mantenerlo visible en la red. Kamailio incluye un módulo llamado RTPPROXY que permite, utilizando una aplicación de rtpproxy, mitigar los problemas que generan los flujos de mensajes RTP al establecer sesiones con SIP.
- **REGISTRAR**: Este módulo tiene la lógica de procesamiento de mensajes REGISTER, para lograr implementar al servidor como un servidor REGISTRAR. Para obtener esto es necesario la inclusión de ciertos módulos y si se está usando a Kamailo como proxy SIP y para la registración de usuarios SIP es necesario que los servidores encargados de la parte de multimedia reciban esta información de registro.
- **TLS**: Este módulo permite el soporte de TLS (Transport Layer Security) para el flujo de mensajes mediante la librería OPENSSL.

[2] http://kamailio.org/docs/modules/4.3.x/

- **TM**: Este módulo transforma a Kamailio en un proxy con estado, ello, debido a que el proxy se encargará de guardar el estado de las transacciones SIP. La importancia de esto estriba en la posibilidad del proxy de determinar si recibe respuesta y en caso negativo, reenviar los mensajes. Hay características importantes que da un proxy con estado sobre uno sin estado a costa de consumos de memoria y CPU. Por ejemplo, una de ellas es proveer la capacidad de reenvíos de peticiones en caso de errores sin recurrir a los clientes.

Ahora bien, Kamailio es un elemento clave en una infraestructura en la cual el servicio de telefonía basado en el protocolo de señalización SIP recibe flujos muy altos y en que plataformas de comunicación como Asterisk y FreeSWITCH como referente principal para la recepción de estos flujos, afectarían altamente el servicio. Por ello, colocar a sistemas similares a Kamailio como referente en la recepción del flujo y mediante la capacidad de hacerse cargo del registro de usuarios, así como de aumentar la escalabilidad para el flujo de llamadas mediante módulos de balanceo de carga para distribuir el flujo a plataformas de comunicación como Asterisk y FreeSWITCH, parece ser una solución adecuada para las necesidades.

Sin embargo, lo anteriormente dicho tiene un pequeño problema a tomar en cuenta, la confiabilidad de tener un único servidor de Kamailio significa un único punto de falla y la caída del servicio debido a un mal funcionamiento a nivel de hardware o de software en este servidor es muy probable. Por ello, se explicarán en las próximas secciones algunas herramientas que permiten el desarrollo de esquemas de alta disponibilidad para proveer redundancia.

6.3 Corosync

Corosync es una de las herramientas para lograr la creación y gestión de un cluster, sin embargo, otras son necesarias para alcanzar el objetivo. A lo largo de esta sección se verá en qué consiste Corosync, su arquitectura y como cohesiona con los distintos componentes que forman un cluster.

6.3.1 Definición

Corosync es una capa de comunicación para la gestión de clusters software libre. Fue creado para simplificar: afincándose sólo en la parte de comunicación cuando esquemas de alta disponibilidad diseñados desde la implementación de OpenAIS eran consideradas muy pesadas y complejas en soluciones de clustering en software libre. OpenAIS es una implementación software libre que sigue las especificaciones de Application Interfaces Specification (AIS) y AIS es un estándar para el desarrollo de interfaces entre el middleware y aplicaciones de servicio propuesto por el SAForum.

6.3.2 Arquitectura de Corosync

Como fue precisado en la definición de Corosync, éste fue creado como un modelo más simple, el cual logra disminuir problemas de interoperabilidad entre distintas herramientas “separando el núcleo de la infraestructura de los servicios del cluster. Al hacer esta abstracción, todos los servicios del cluster pueden cooperar en la toma de decisiones en el cluster” [20].

La arquitectura que forma el motor de Corosync se ve en la Figura 6.3 tomada de [20]:

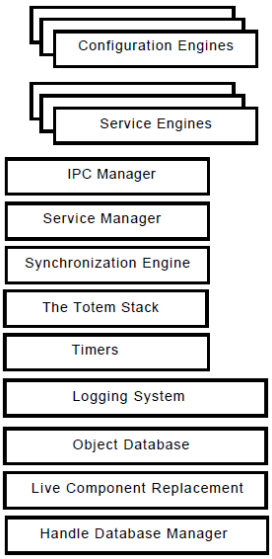


Figura 6.3: Arquitectura de Corosync

El motor de Corosync está compuesto por múltiples componentes dentro de su arquitectura (gestor IPC, gestor de servicios, etc). A lo largo de la especificación de la arquitectura de Corosync, se explicarán algunos de los componentes que forman parte de ella, sobre todo aquellos que tienen un impacto directo en la lógica acerca de cómo Corosync ofrece y logra su finalidad.

Timer

Cualquier service manager (uno de los componentes que se hablará más adelante) requiere el uso de timers, ya que estos permiten gestionar los tiempos en que se requiera aplicar alguna acción específica que se requiera de los servicios [20].

Totem Stack

El Totem Stack cuenta con dos componentes, el Totem Single Ring Ordering y el Membership Protocol que implementa un modelo de comunicación síncrona virtual (extended virtual synchrony communication model). Este modelo consta de múltiples propiedades, una de estas, es la capacidad de soportar operaciones concurrentes mediante intercambios de tokens en una infraestructura de anillo impuesta y enviar de manera ordenada los mensajes requeridos. Lo que permite mantener sincronizada la información del estado del cluster en los nodos [20].

Ahora bien, la primera propiedad no es identificable sin que, mediante el Membership Protocol, el cluster identifique la falla de algún nodo, particionamiento de redes o pérdida de todos los tokens del cluster. En caso de que ocurra, el protocolo se encarga de reconstruir un nuevo anillo para que se reinicien las operaciones. Con todo esto, el Totem Stack es el que se encarga de verificar y mantener el estado funcional del cluster.

Configuration Engine

El configuration engine es creado para permitir que los archivos de configuración tengan independencia con el motor de Corosync, cargando una aplicación específica de configuration engine.

Interprocess Communication Manager

“El Interprocess Communication Manager es responsable de la recepción y transmisión de peticiones IPC” [20]. La finalidad de este componente es la comunicación entre componentes, tomando peticiones IPC entrantes del service manager al service engine correspondiente. Este gestor es necesario debido a que el service engine usualmente es un componente que no es diseñado por Corosync.

Service Engine

“Un service engine es creado por terceros para proveer alguna forma de gestión de servicios por parte del cluster” [20], ejemplos de estos son Pacemaker y CMAN (Cluster Manager). Este componente tiene una interfaz que permite enlazar en tiempo real con el service manager para mantener la comunicación entre Corosync y el service engine escogido [20].

Service Manager

“El service manager es responsable de cargar y descargar los plugins de service engine. También es responsable de enrutar las peticiones a los service engines cargados en el cluster de Corosync” [20].

Es importante saber cómo los componentes se enlazan entre sí mediante el service manager: cuando el motor de Corosync inicializa, el configuration engine es cargado. De los archivos de configuración se obtienen los service engine a cargar por el service manager y finalmente el service manager carga todos los service engines.

Una vez el service manager se encargue de cargar todos los servicios y los service engine correspondientes, cuando el usuario haga una petición vía el interprocess communication manager, el componente lo enruta al service engine apropiado mediante el service manager. El service manager también se encarga de enviar los cambios dentro del cluster según lo reportado por el Membership Protocol a los service engine. Luego, un service engine replica la información relacionada de los cambios vía el Totem Single Ring Protocol. Estos mensajes transmitidos son gestionados y enrutados por el service manager a otro service engine. Finalmente el service manager

es responsable de enrutar las actividades de sincronización mediante el synchronization engine en caso de haberse reportado algún cambio por parte del Membership Protocol [20].

Synchronization Engine

El synchronization engine es responsable de gestionar la recuperación de los service engines luego de una falla o agregación/eliminación de algún nodo. Un service engine puede opcionalmente utilizar o no el synchronization engine [20].

Ya se tiene la herramienta que se encarga de la comunicación y de los permisos en el cluster. Faltan otras herramientas que permitan la gestión de los servicios en el cluster, de esto se encarga Pacemaker. Cuando se haga referencia Pacemaker se explicará cómo trabajan ambos entre sí para lograr la solución de clustering deseada.

6.4 PaceMaker

En las secciones previas se vieron distintas herramientas que proveen la capacidad de ofrecer servicios de VoIP de manera óptima dentro de una infraestructura con una gran capacidad de integración de mecanismos para ofrecer disponibilidad y escalabilidad. Sin embargo, estas no son suficientes para la creación de una infraestructura de clustering. Para ello se necesitan herramientas que logren gestionar los nodos y los servicios que ofrece la infraestructura, Pacemaker ofrece esta capacidad.

6.4.1 Definición

Pacemaker es un Cluster Resource Manager (CRM) que se encarga de "…lograr la máxima disponibilidad para los servicios (recursos) de tu cluster, detectando y recuperando al cluster de fallas a nivel de los nodos y los recursos haciendo uso de las capacidades de mensajería y permisologias que provean tu infraestructura de cluster preferida (Corosync o Heartbeat)" [21] .

Entre algunas de las características de Pacemaker, según se especifica en [21] se encuentran:

- Detección y recuperación en caso de fallas, tanto a nivel de nodos como de servicios.
- Abstracción de los recursos. Una vez configurado el recurso, Pacemaker no requiere saber de qué recurso se trata.
- Soporta la integridad de información utilizando un módulo llamado STONITH.
- Capacidad de especificar el orden de los recursos del cluster.
- Soporte de tipos avanzados de servicios.
    - Clones: servicios que pueden estar activados en varios nodos.
    - Multi-estados: servicios con múltiples modos.

La manera en que logra ofrecer estas características es mediante una arquitectura de componentes internos de Pacemaker y con herramientas de comunicación y gestión de

permisologias como Corosync o Heartbeat.

6.4.2 Arquitectura de Pacemaker

Stack de Pacemaker

La arquitectura, desde el punto de vista macro, consta de tres áreas siguiendo lo explicado en [21]:

- Componentes no conscientes del cluster: Se trata de componentes como los recursos, los scripts de comienzo paro y monitoreo de ellos; y demonios locales.
- Gestor de Recursos: Componentes capaces de reaccionar a eventos que sucedan en el cluster que gestiona. Eventos, tanto a nivel de nodos como de recursos. Algunos de estos eventos pueden ser: incluir o quitar nodos dentro del cluster, fallas en algún recurso, y obtener respuestas de parte del componente como mover recursos a otros nodos, parar nodos, etc.
- Infraestructura a bajo nivel: A este nivel es donde proyectos como Corosync, Heartbeat o CMAN hacen su trabajo, encargándose de la comunicación entre nodos del cluster para verificar su estado y gestión de permisologias.

En la Figura 6.4 tomada de [21] se puede ver el stack de Pacemaker. Los componentes que se encuentran en color amarillo son sistemas de archivos para cluster (cLVM2, GFS2 y OCFS2). Estos tienen como finalidad facilitar los problemas que se originan en el acceso compartido a la información dentro de ese sistema de archivos. Por su parte el DLM (Distributed Lock Manager) es un gestor utilizado para manejar el acceso a esos sistemas de archivos, evitando la corrupción del mismo. Los componentes que se encuentran en color verde son utilizados para la gestión de los recursos en el cluster y el componente en rojo, en este caso marcado como Corosync, ya ha sido descrito anteriormente.

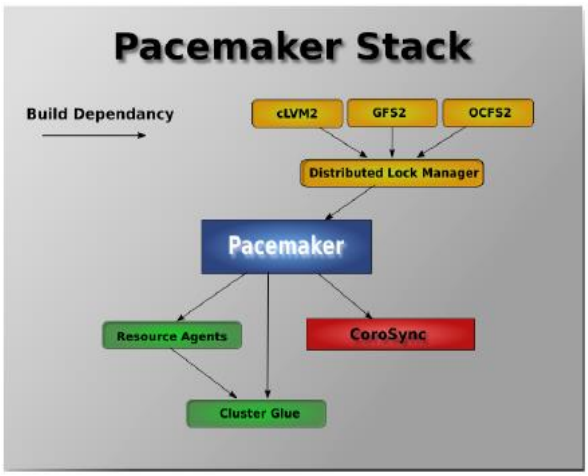


Figura 6.4: Estructura del Stack de Pacemaker

Componentes Internos

Pacemaker consta de 5 componentes claves:

- Cluster Information Base (CIB).
- Cluster Resource Management daemon (CRMd).
- Local Resource Management daemon (LRMd).
- Policy Engine (PEngine o PE).
- Fencing daemon (STONITHd).

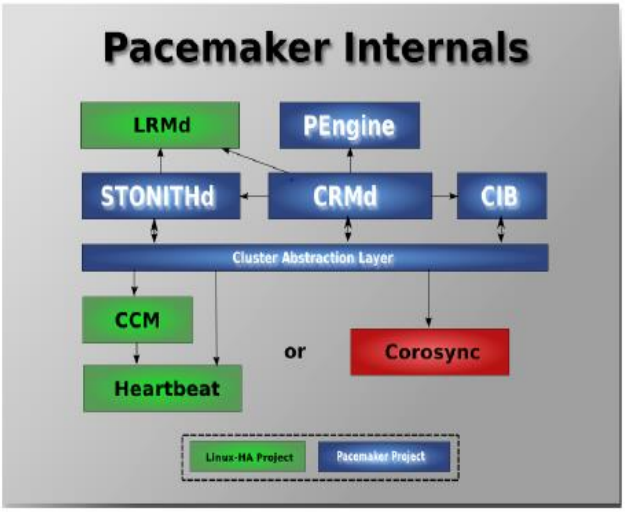


Figura 6.5: Diagrama de Componentes Internos que Forman Pacemaker

En la Figura 6.5 tomada de [21], CIB es el componente encargado de almacenar las configuraciones del cluster y almacenar los estados recientes de los recursos del cluster. Estas configuraciones, y los estados de los recursos, están en formato XML. “El contenido del CIB es mantenido sincronizado automáticamente alrededor del cluster entero y son usados por el PEngine para computar el estado ideal del cluster y como debería ser logrado” [21].

En Pacemaker cada nodo perteneciente al cluster debe tener CRMd y entre estos se

escoge uno, el cual actuará como maestro llamado DC (Designated Controller). El DC se encargará asumir las instrucciones mandadas por PEngine en el orden requerido y luego pasarlas al LRMd del nodo elegido como DC o bien a los CRMd de los nodos pertenecientes al cluster. Ello, mediante la infraestructura de comunicación de cluster escogido (Corosync o Heartbeat), para después ser pasada por sus LRMd's respectivos con miras a su procesamiento.

"Los nodos reportan el resultado de sus operaciones de vuelta al DC y, apoyado en los resultados a que se aspira y los actuales, ejecutarán cualquier acción que necesitará esperar por la previa para ser completada, o abortar el procesamiento y preguntar por el PEngine, para recalcular el estado ideal del cluster, basados en el resultado inesperado" [21].

Tipos de Clusters de Pacemaker

Pacemaker soporta múltiples técnicas de alta disponibilidad, activo-activo, activo-pasivo, N a 1, N más 1, entre otros.

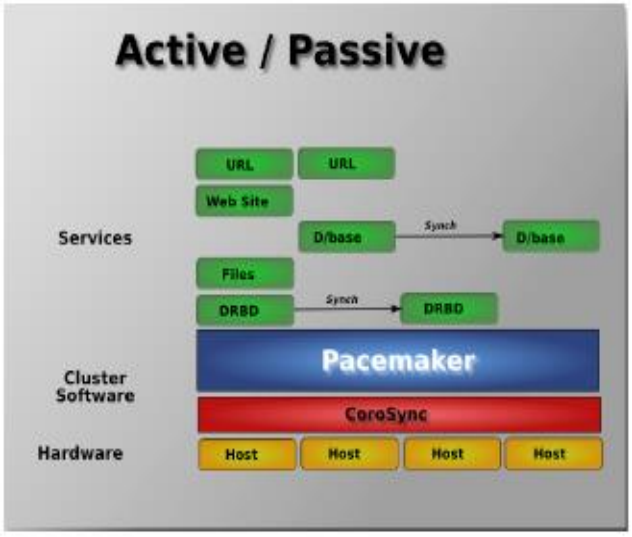


Figura 6.6: Esquema Activo-Pasivo con Pacemaker y Corosync

La Figura 6.6 tomada de [21] muestra una solución de cluster activo-pasivo, ya explicada anteriormente en este trabajo. Se destaca en la Figura 6.6 el uso de herramientas como Pacemaker, Corosync y DRBD. DRBD es utilizado para sincronizar y mantener al nodo pasivo actualizado, en cuanto a la información se refiere, del nodo activo.

La Figura 6.10 tomada de [21] explica una solución activo-activo e incluso cualquier solución N más M, su arquitectura depende de cómo se quiera implementar.

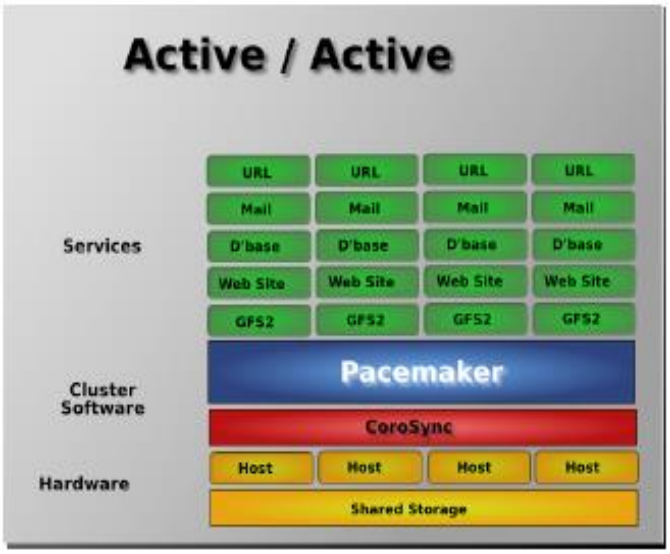


Figura 6.7: Esquema Activo-Activo o N más N con Pacemaker y Corosync

6.4.3 Recursos del Cluster

Una vez comentado lo concerniente a los nodos del cluster y a algunas de las ventajas que ofrece la herramienta, es importante recalcar que es un recurso y sus características en Pacemaker.

Definición

"Un recurso es un servicio hecho altamente disponible por un cluster" [21]. Pacemaker define distintos tipos de recursos, los cuales son:

- **Primitivo**: este es el recurso más común y simple configurado en Pacemaker.
- **Grupo**: Un grupo es una pluralidad de recursos primitivos que requieren ser gestionados en conjunto, por ejemplo, migración de recursos, fallas en el recurso, etc.
- **Clones**: Este tipo de recurso consiste en múltiples instancias de un mismo servicio distribuidos a lo largo de los nodos de un cluster, técnica muy útil para esquemas de balanceo de cargas.
- **Multi-Estado**: Este tipo de recurso consiste en un tipo de recursos clon, el cual permite que las instancias estén en uno de los dos modos de operación (llamado roles). "Los roles son llamados Maestro y Esclavo, pero pueden significar lo que se quiera que signifiquen. La única limitación es que cuando una instancia sea comenzada, tendrá el rol de Esclavo" [21].

Cada recurso primitivo tiene un agente de recursos. Este agente es un programa que permite al cluster abstraerse del recurso propiamente, ya que confía en la gestión que hace el agente de recursos, que es la de abstraer el servicio y adecuarlo para ser compatible en la estructura del cluster. Ahora, hay distintos tipos de agentes, como se

verá más adelante.

Clases de Recursos

Hay distintas clases de agentes de recursos, pero no todas serán abarcadas en este trabajo. Entre ellos se destacan los siguientes:

- **Open Cluster Framework (OCF):** “Los estándares de OCF son básicamente una extensión de las convenciones de LSB para scripts de inicio con el objeto de soportar parámetros, hacerlos descriptibles por sí mismos y hacerlos extensibles” [21]. Los parámetros son pasados hacia el agente mediante variables de entorno con el prefijo “OCF_RESKEY_”. El recurso le pasa el parámetro al agente con el nombre que resulte del prefijo concatenado con la variable.
- **Linux Standard Base (LSB):** Los agentes de recurso LSB son usualmente conseguidos en /etc/init.d en Linux. Estos son los típicos scripts de gestión de inicio o culminación de servicios en los distintos sistemas operativos Linux. “Muchas distribuciones de sistemas operativos afirman cumplir con los requerimientos de LSB, pero terminan siendo scripts de inicio incompatibles con LSB” [21].
- **Systemd.**
- **Upstart.**
- **STONITH.**
- **Plugins.**

Operación de Recursos

Las operaciones son acciones que el cluster aplica sobre los recursos. Entre ellos destaca empezar, parar o monitorear el recurso. Por defecto el cluster no se encarga de asegurar si un recurso se encuentra en buen estado, pero se puede encargar agregando una operación “monitor” en la definición del recurso especificando intervalos de tiempo. Este tiempo es configurable en las propiedades de una operación definida, además de tomar decisiones en caso de falla del recurso entre esos intervalos como, por ejemplo, el reinicio del recurso, apagar el nodo, etc.

Ahora, Pacemaker no solo se encarga de administrar los recursos cuando están activos y vigilar su estado, sino también se encarga de identificar en cuáles nodos ciertos recursos correrán como también en el orden en que se ejecutarán, conforme se verá más adelante.

6.4.4 Recursos Del Cluster

Hay distintas decisiones que permite Pacemaker para optimizar y asegurar el funcionamiento del cluster, algunas de estas se verán a lo largo de esta sección.

Decidir en qué Nodo un Recurso puede ser Corrido

Se puede crear una restricción en que nodos un recurso no puede correr o viceversa.

Orden de Recursos y Dependencias

La manera de especificar el orden en que los recursos deben iniciar es creando una restricción (rsc_order). Una vez especificado el rsc_order se utilizan ciertas propiedades para elegir el orden. Algunas de las propiedades según lo especificado en [21] son:

- **first**: Contendrá el nombre del recurso que inicia antes del especificado en el parámetro la propiedad **then**.
- **then**: Nombre del recurso que debe correr luego del especificado con la propiedad **first**.
- **kind**: Utilizado para forzar una restricción, esta tiene distintos valores:
  - **optional**: esta restricción se vuelve opcional, lo que indica que la restricción especificada con los parámetros **first** y **then** es aplicada solo cuando se inicia o se destruye el cluster y no cuando hay algún cambio de estado en el recurso especificado en **first**.
  - **mandatory**: En este caso la restricción siempre debe ser tomada en cuenta en ambos casos (los especificados en optional).
  - **serialize**: Mayormente usado para asegurar que ambos recursos arranquen o paren en el momento adecuado y que ambos logren hacerlo sin problemas

La ordenación de recursos puede ser una cadena de servicios como se verá en la Figura 6.8 tomada de [21]:

```
1: <constraints>
2: <rsc_order_ id=”order-1” first=”A” then=”B”/>
3: <rsc_order_ id=”order-2” first=”B” then=”C”/>
4: <rsc_order_ id=”order-1” first=”C” then=”D”/>
5: </constraints>
```

Figura 6.8: Ejemplo de Configuración de Ordenamiento de una Cadena de Recursos

Esto significa que al aplicar alguna acción de comienzo/detención de los recursos del cluster especificados en esa restricción, deberán ejecutarse en ese orden, primero se aplica la acción en el recurso A, luego en el B, y así sucesivamente.

6.4.5 STONITH

STONITH (Shoot the Other Node in the Head) se encarga de “…proteger los datos

contra la corrupción por parte de nodos rebeldes o de acceso concurrente” [21].

La existencia de STONITH previene el mal funcionamiento del cluster y corrupción de datos en escenarios donde no se conozca de manera fidedigna el estado de un nodo por un periodo de tiempo, en estos casos, STONITH toma medidas preventivas de aislamiento de esta entidad para proteger la integridad de los datos y del cluster.

Ahora, ¿Qué sucede con el almacenamiento? En relación con esta pregunta, se puede destacar, que los sistemas de archivos comunes no pueden ser montados ni tener acceso por múltiples dispositivos a la vez. Además, estaría la necesidad de tener un almacenamiento compartido para este tipo de soluciones como NFS (Network File System), NAS (Network Attached Storage), SAN, entre otros. Por esto, como se explicará en la siguiente sección, existe DRBD para la replicación de datos y permite que existan soluciones de alta disponibilidad como activo-pasivo o activo-activo sin compartir el almacenamiento.

6.5 DRBD

La creación de una solución de clustering no solo depende de mantener recursos que proveen el servicio altamente disponible y con redundancia sino también de proveer mecanismos para que los datos e información se mantengan redundantes y en la capacidad de proveer los datos en escenarios de contingencia.

Mecanismos para ofrecer almacenamientos compartidos y de replicación síncrona de datos son importantes e imprescindibles para ofrecer soluciones altamente disponibles.

6.5.1 Definición

El Distributed Replicated Block Device (DRBD) es una herramienta de software que ofrece una solución de replicación en tiempo real y transparente de almacenamiento de dispositivos como discos duros, particiones, etc, entre servidores [22].

“Las funcionalidades del núcleo de DRBD son implementadas de manera parecida a un módulo de kernel de Linux. Específicamente, DRBD constituye un driver para un dispositivo de bloque virtual…” [22], por ello DRBD se encuentra muy al fondo de la pila de I/O del sistema. Debido a esto, DRBD es independiente de casi cualquier aplicación que se encuentre en el sistema para proveer sus funcionalidades de replicación y proporcionando alta disponibilidad. La ubicación del kernel de DRBD puede verse en la Figura 6.9 tomada de [22].

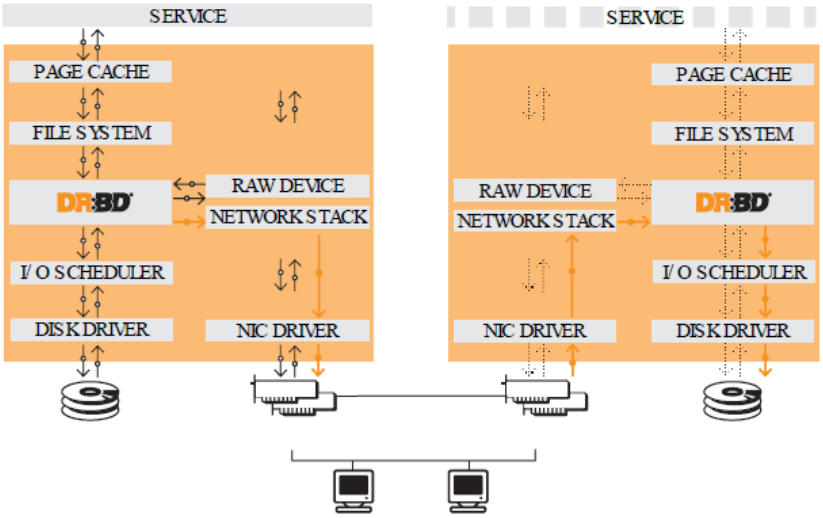


Figura 6.9: Posición de DRBD en la Pila de I/O de Linux

6.5.2 Recursos en DRBD

Un recurso DRBD es un conjunto de aspectos que forman parte de un dispositivo de almacenamiento replicado por DRBD. Entre ellos están:

- **Nombre de Recurso**: Como se indica, es el nombre que identifica al recurso.
- **Dispositivo DRBD**: Es el dispositivo de bloque virtual que está siendo manejado por DRBD, usualmente es representado como /dev/drbd[0-9]+.
- **Configuración de Disco**: Esto representa la información que es copiada y la meta data usada por DRBD.
- **Configuración de Red**: Representa todos los aspectos de comunicación de DRBD con el o los otros recursos que se encuentre gestionando.

Ahora, cada recurso tiene un rol, que puede ser Primario o Secundario. El uso de primario y secundario es muy parecido al de activo y pasivo en técnicas de alta disponibilidad de clustering. Por ello, se utilizó el de primario y secundario para diferenciarlos. El recurso en rol primario es utilizado sin restricción, tanto para operaciones de lectura como de escritura.

El recurso en rol secundario recibe las actualizaciones de información por parte del recurso o nodo en rol primario dentro de la solución de replicación de DRBD. Pero éste no puede ser accedido por aplicaciones, tanto para lectura como para escritura. “La razón por la cual se restringe incluso el acceso por lectura al dispositivo es la necesidad de mantener coherencia con cache, la cual sería imposible si el recurso en rol secundario tuviese permitido el acceso a lectura” [22].

Habiendo aclarado lo que es un recurso y los roles que puede tomar dentro de un cluster, es importante conocer las capacidades que tiene DRBD de proporcionar la replicación de información de manera óptima.

6.5.3 Esquemas de Replicación

En DRBD existen dos esquemas de replicación que pueden ser utilizados en múltiples soluciones topológicas de clustering. Estos esquemas son:

Modo Único – Primario

En este esquema siempre existe un único nodo en rol primario en el cluster. “Cómo se garantiza que un único nodo en el cluster manipulará la información en cualquier momento, este modo puede ser utilizado con sistemas de archivos convencionales (ext3, ext4, XFS, entre otros)” [22]. Este esquema es el que se usaría en soluciones de alta disponibilidad de failover activo-pasivo.

Modo Dual – Primario

En este esquema, ambos nodos pueden estar en rol primario, lo que significa que ambos podrán tener acceso a la información. Por ello, si se está utilizando este esquema, sistemas de archivos comunes como ext4, ext3 o XFS no pueden ser empleados. Para este tipo de esquemas hay que usar sistemas de archivos que utilicen el Distributed Lock Manager (DLM) que son para acceso compartidos como en clusters. Más adelante en esta sección se explicará lo que es el DLM y algunos de estos sistemas de archivos.

Una vez explicados los distintos esquemas de replicación, es de importancia tener claro que existen distintos protocolos de replicación. La variedad de protocolos es relevante, ya que algunas ofrecen mayor confiabilidad y protección en la coherencia con los datos de ambos nodos a cuesta de latencia. Los protocolos son los siguientes:

Protocolo de Replicación Asíncrona

En este protocolo “las operaciones de escritura locales en el nodo primario son consideradas completas cuando la escritura en el disco local haya ocurrido y el paquete de replicación ha sido colocado en el buffer de envío de TCP local” [22]. En pocas palabras los datos no han llegado al nodo secundario, lo que provoca que, en caso de tener que aplicar el failover forzado, algunos de los datos se podrían perder.

Protocolo de Replicación de Memoria Síncrona

En este caso las operaciones de escritura en el nodo primario son consideradas completas cuando además de haberse escrito los cambios en el disco local del nodo primario, éstas hayan sido enviadas y recibidas por el nodo secundario. A diferencia del protocolo explicado anteriormente, no hay pérdida de datos al aplicarse failover. Sin embargo, si hay fallas de poder en ambos nodos simultáneamente o problemas irreversibles en los datos del nodo primario, los cambios más recientes aplicados en el nodo primario se perderán [22].

Protocolo de Replicación Síncrona

Este protocolo es el que ofrece mayor protección y confiabilidad ya que las operaciones de escritura en el nodo primario son consideradas completas cuando es confirmada la escritura tanto en el disco local como en el disco del nodo secundario. Esto ofrece mayor protección y asegura que no se perderán datos al aplicarse failover. Sin embargo, siempre hay probabilidad de que se pierdan datos en casos como, corrupción del sistema de archivo en alguno de los nodos, o si ambos nodos sufren una caída por falla de electricidad [22].

6.5.4 Sistemas de Archivo para Clustering

Los sistemas de archivos comunes como ext4, NTFS, XFS, entre otros, son sistemas de archivos que llevan años en el mercado y que son óptimos para el control y gestión de los archivos. Sin embargo, sólo pueden ser montados y accedidos por un único sistema. Debido a esto, nacen los sistemas de archivos para clustering, ya que múltiples soluciones de clustering que ofrecen mayor disponibilidad y desempeño requieren compartir los datos que son, en su mayoría, encontrados en el sistema de archivos.

Existen múltiples sistemas de archivos para clustering según los niveles de acceso que se permiten. Por ello, se tocarán solamente aquellos sistemas de archivos que proveen acceso compartido a nivel de bloques y que usa SAN para proveer acceso a discos por múltiples computadores. Estos sistemas de archivos son llamados sistemas de archivos de disco compartido. Además de lo explicado anteriormente, es importante tener claro que esto se logra con el kernel de cada sistema gestionando la sincronización y el acceso al sistema de archivos. También utiliza un modelo que permite la gestión al acceso de los archivos Este modelo se llama Distributed Lock Manager (DLM).

Distributed Lock Manager

DLM es un recurso que corre en cada nodo del cluster el cual tiene múltiples funcionalidades. El sistema de archivo GFS2 (Global File System 2) utiliza el DLM para sincronizar el acceso a los archivos y a la meta data, en cambio CLVM utiliza DLM para sincronizar las actualizaciones de los volúmenes LVM (Logical Volume Manager) y grupos de volúmenes[3].

El modelo DLM provee múltiples modos de bloqueo, los cuales incluyen ejecuciones tanto síncronos como asíncronos. El DLM se encarga de rechazar el acceso a recursos de bloqueo. Estos recursos de bloqueo constituyen aquellos datos a los cuales se les desea bloquear. Ellos pueden contener toda una base de datos como un solo archivo de esa base datos, lo que implica la flexibilidad y la granularidad con la que se puede configurar.

[3] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/High_Availability_Add-On_Overview/ch-dlm.html

“El DLM provee sus propios mecanismos para soportar las características de bloqueo, como la comunicación entre nodos para manejar el tráfico de bloqueo y protocolos de recuperación para gestionar los bloqueos luego de una caída de un nodo o la migración de bloqueos cuando un nuevo nodo se une al cluster”[3].

Ahora, teniendo claro, lo que es DLM y su funcionalidad, quedan por explicar algunos de los más importantes sistemas de archivos de disco compartidos que utilizan DLM como gestor bloqueos y acceso a datos.

OCFS2

Oracle Cluster File System 2 (OCFS2) es un sistema de archivo de disco compartido para Linux con el objeto de proporcionar alto rendimiento y alta disponibilidad. Este sistema de archivo fue creado por Oracle y era ampliamente usado en soluciones de Oracle RAC y base de datos Oracle. En el 2006, fue completamente integrado en los kernels de Linux y se encuentra disponible en la mayoría de las distribuciones [23].

OCFS2 provee soluciones importantes en esquemas de disco compartido, “aplicaciones de clustering toman ventaja de la funcionalidad de coherencia en cache y I/O en paralelo por más de un nodo a la vez para proveer mayor rendimiento y escalabilidad” [23]. Además, se apoya del recurso DLM, el cual gestiona el acceso concurrente a los nodos del cluster para proveer el bloqueo al acceso a datos y lograr mantener la consistencia de los datos.

Ejemplos de cómo OCFS2 ofrece consistencia de datos es manteniendo los Inodos intactos hasta que dejen de ser utilizados por el resto del cluster. Si un nodo decide eliminar un archivo mientras está siendo utilizado por otro nodo del cluster, el archivo es eliminado, pero el Inodo se mantiene hasta que deje de ser utilizado por el resto de los nodos del cluster, en el momento que culmine su uso será eliminado.

GFS2

Global File System 2 (GFS2) “es un sistema de archivos simétrico de 64 bits para cluster, el cual es derivado del anterior sistema de archivo GFS. Es diseñado principalmente para aplicaciones SAN en el cual cada nodo en el cluster tiene acceso igualitario al almacenamiento” [24].

Igual que lo anteriormente explicado en OCFS2, para manejar la integridad del sistema de archivo se utiliza DLM para la gestión de bloqueos y acceso a los datos. La manera en que GFS2 se complementa con DLM para lograr el bloqueo de los bloques de datos es dividiendo los bloqueos requeridos por el cluster, el cual GFS2 los llama glocks. En diferentes tipos de bloqueo, la concatenación del número de glock y el tipo de glock es enviado a DLM para que el gestione el acceso a ese espacio bloqueado.

6.6 SIPp

Las herramientas explicadas hasta este momento permiten la implementación de una infraestructura de telefonía de alta disponibilidad y escalable, pero ¿cómo se hace para

demostrar su confiabilidad y disponibilidad ante flujos altos de llamadas?

SIPp es la respuesta a la propuesta en este trabajo debido a su posibilidad de generar altos flujos de establecimiento de sesiones, con la capacidad de definir distintos escenarios de prueba y utilizar las métricas óptimas para evaluar a los componentes de la infraestructura en flujos de mensajes SIP. Con esto, se puede llegar a una conclusión de las soluciones de infraestructura implementadas y definir así si cumple con las exigencias de un sistema de alta disponibilidad robusto.

6.6.1 Definición

SIPp es una herramienta utilizada para realizar pruebas de desempeño de sistemas que utilicen SIP como protocolo de señalización. Permite la simulación de múltiples UAs tanto UAC como UAS estableciendo llamadas mediante métodos INVITE y culminándolos con BYE. Utiliza archivos XML para la descripción de los escenarios de simulación, los cuales pueden ser modificados dinámicamente utilizando expresiones regulares y variables para mayor flexibilidad en la configuración de los escenarios [25].

6.6.2 Funcionalidades

Algunas de las funcionalidades de SIPp son:

Soporte de UAS y UAC

SIPp tiene múltiples escenarios que son soportados, además tiene la ventaja de que, aunque SIPp da la posibilidad de utilizar escenarios personalizados, incluye algunos escenarios por defecto. Entre los escenarios por defecto, como especificado en [25], soportados por SIPp están:

- **UAC**: Este sería el escenario más común, donde SIPp se encarga es de generar flujos de establecimiento de sesiones enviando mensajes INVITE y donde ocurre el procedimiento común de establecimiento de sesión, recibiendo mensajes TRYING y RINGING hasta recibir el OK y culminar el establecimiento con ACK.

  Una vez culminado, no hay flujo de mensajes hasta terminar la sesión con BYE. Este procedimiento es explicado, según lo especificado en [25], en la Figura 6.10.

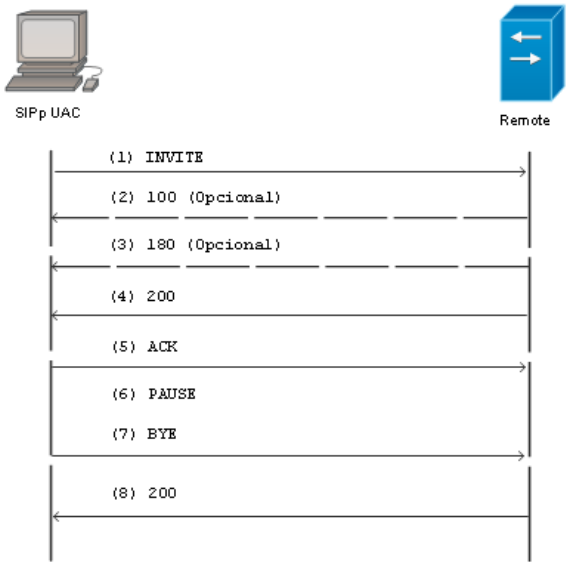


Figura 6.10: Flujo de Comunicación de SIPp como UAC

- **UAC con Media**: Es el mismo escenario que el de UAC pero en lugar de pausar el flujo al establecer la sesión, hay un intercambio de mensajes RTP como flujo de audio. Este flujo de audio es un archivo cargado y adjuntado al archivo XML que establece el escenario.
- **UAS**: El intercambio de mensajes es el mismo que el planteado en UAC pero los entes se intercambian. El ente remoto es el que empieza con INVITE y SIPp es el que acepta la sesión, el flujo puede verse, según lo especificado en [25], en la Figura 6.11.

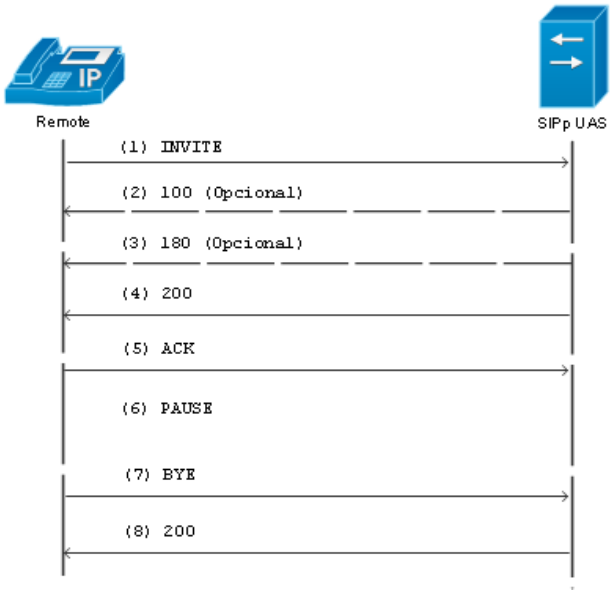


Figura 6.11: Flujo de Comunicación de SIPp como UAS

- **3PCC**: Es utilizado para permitir a una instancia de SIPp a hablar con distintos dispositivos remotos a la vez.

Creación de Escenarios Personalizados

SIPp permite crear escenarios personalizados en formato XML. La Figura 6.12, según lo especificado en [25], muestra como debe ser el encabezado simple del escenario.

```
1: <?xml version="1.0" encoding="ISO-8859-1" ?>
2: <scenario name="Basic Sipstone UAC">
3: </scenario> (cierra la configuración del escenario)
```

Figura 6.12: Encabezado de un Escenario SIPp

Los escenarios tienen distintos elementos que parten, claramente, de una configuración de XML. Algunos de estos son los atributos, comandos (el nombre de la etiqueta) y el cuerpo del mensaje.

Ahora, hay atributos que son únicos para un comando y otros que son universales. La Tabla 6.1, según lo especificado en [25], muestra algunos de estos atributos.

Atributo	Descripción	Ejemplo

Next	Permite ir a otra parte del script al culminar una acción (usualmente un envío de mensaje u opcionalmente el recibimiento de un mensaje)	Saltar a la etiqueta “12” después de enviar un mensaje ACK
		`1: <send next=”12”>`
		`2: <![CDATA[`
		`3: ACK sip:[service]@[remote_ip]:[remote_port]`
		`4: SIP/2.0`
		`5: …`
		Figura 6.13: Ejemplo de Uso del Atributo Next

Test	El atributo “test” es utilizado en conjunto con el atributo “next”. Cuando el comando donde el atributo “next” es ejecutado, debe cumplirse la condición impuesta en el comando “test”, si ésta no es cumplida la acción del atributo “next” no es ejecutado.	Saltar a la etiqueta “6” luego de enviar un mensaje ACK, solo si la variable 4 es colocada
		`1: <send next=”6” test=”4”>`
		`2: <![CDATA[`
		`3: ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0`
		`4: …`
		Figura 6.14: Ejemplo de Atributo Test

Counter	Se incrementa el contador puesto como parámetro cuando un mensaje es enviado, luego es enviado al archivo de estadísticas	*<send counter=”MsgA”>*: El contador “MsgA” es incrementado cuando un mensaje es enviado

Tabla 6.1: Atributos Utilizables por Cualquier Comando SIP

Los atributos incluyen funcionalidades para cada comando, pero los comandos son los que les otorgan la característica y el comportamiento a los escenarios. Varios de los comandos más importantes están descritos la Tabla 6.2, según lo especificado en [25].

Comando	Descripción	Ejemplo

<send>	Comando que representa el envío de un mensaje. El mensaje a enviar está definido dentro de un CDATA, en donde se debe especificar las cabeceras del mensaje a enviar y su contenido.	Definición de un mensaje a envíar 1: <send > 2: <![CDATA[3: INVITE: sip:[service]@[remote_ip]:[remote_port] SIP/2.0 4: Via: SIP/2.0 [transport] [local_ip]:[local_port];branch=[branch] 5: … **Figura 6.15:** Ejemplo de Uso del Comando Send
<recv>	Comando que representa la recepción de un mensaje. Debido a la naturaleza de SIPp y por cuestiones de estadísticas, tiene atributos propios que modifican el comportamiento deseado. Por ejemplo, la definición de recepción de un tipo de mensaje en específico, en caso de recibir un mensaje de otro tipo se recibe como un mensaje no deseado.	SIPp esperará la recepción de un mensaje ACK: *<recv request="ACK">*
<pause>	Comando que representa un periodo de pausa en el flujo de mensajes, este comando es muy utilizado cuando no se está también enviando flujo multimedia sino solamente mensajes SIP.	Especificando la pausa del flujo en 5 segundos: *<pause milliseconds="5000">*
<nop>	Comando utilizado para realizar alguna acción que se requiera (especificada con el comando <action>), no se encarga de realizar alguna acción a nivel de SIP.	Ejecutar play_pcap_audio, atributo que se encarga de producir algún archivo multimedia. 1: <nop> 2: <action> 3: <exec play_pcap_audio="pcap/g711a.pcap/> 4: </action> 5: </nop> F**igura 6.16:** Ejemplo de Uso del Comando Nop
<label>	Comando utilizado para cuando se requiere proseguir a una parte específica de un escenario. Se complementa con atributos como "next" y "test" para lograr la acción deseada.	Colocar un label con número 13: *<label id="13"/>*

Tabla 6.2: Comando más Relevantes en SIPp

Pantallas

SIPp consta de 4 pantallas para monitorear el tráfico en el que se encuentra SIPp, al cual se le puede cambiar de pantalla apretando los hot keys del 1 al 9:

- **Pantalla del Escenario**: Esta pantalla muestra el flujo de llamadas entre los entes y otro tipo de información relacionada a la configuración del escenario, en la Figura 6.17 tomada de [25] se puede ver que esta pantalla muestra información como la tasa de llamadas en el tiempo transcurrido, puerto de destino, tiempo transcurrido, el total de llamadas en ese tiempo, entre otros.

- **Pantalla de Estadísticas**: Esta pantalla se encarga de mostrar estadísticas relacionadas con cantidad y duración de llamadas y tipo de llamadas, tiempos de duración del escenario, entre otros.
- **Pantalla de Repartición**: Esta pantalla muestra la distribución de llamadas respondidas y longitud de llamadas.
- **Pantalla de Variables**: Esta pantalla muestra información de las variables establecidas en el escenario y las acciones por mensaje.

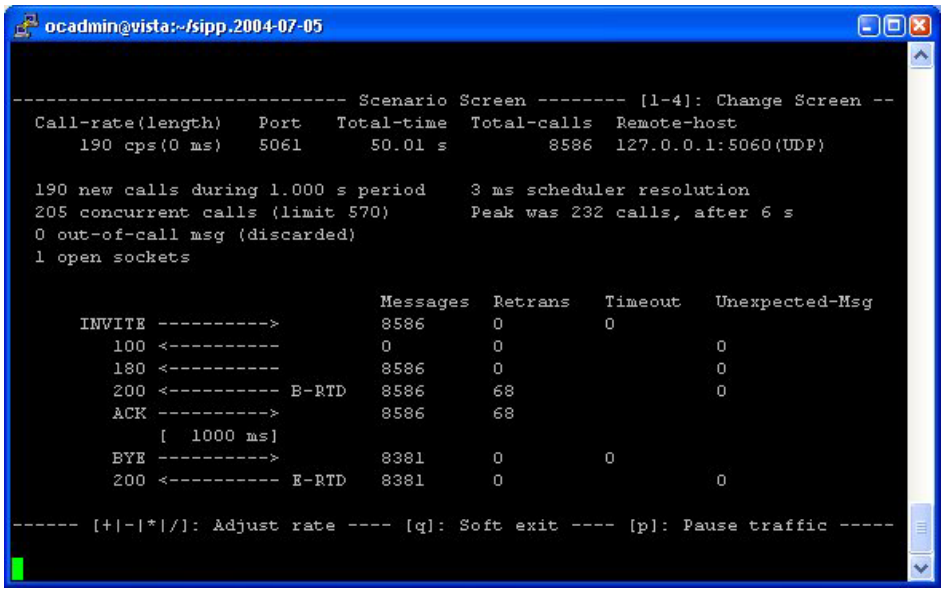


Figura 6.17: Ejemplo de la Visión de la Pantalla de Escenario

Gestión de Media en SIPp

Debido a la naturaleza de SIPp que se encarga principalmente de poner a prueba una infraestructura con el protocolo de SIP, las pruebas multimedia mediante protocolos como RTP son algo escasas, sin embargo, algunas funcionalidades están desarrolladas para su uso en SIPp. Los siguientes puntos indican algunas de estas capacidades:

- **Transmisión RTP**: SIPp permite reproducir archivos de audio que se encuentren codificado en G729, PCMA, o PCMU (archivos .wav) sobre RTP.
- **Reproducción por PCAP**: SIPp permite reproducir sobre RTP archivos .pcap mediante la librería PCAP. Estas transmisiones pueden ser capturadas por herramientas de sniffing como Wireshark o tcpdump. Esta tiene ciertas ventajas, por ejemplo, la transmisión puede estar codificada por cualquier codec debido a que SIPp no se encarga de la codificación como tal y se comporta como una reproducción de una transmisión multimedia por RTP.

Herramientas de prueba que pongan al límite a implementaciones que prestan servicio, y de los que se esperan tener un alto nivel de tráfico es totalmente indispensable. Todo esto debido a que, antes de proveer un servicio, es necesario tener en claro las capacidades de una implementación y definido los niveles de servicio.

Por esto, herramientas como SIPp permiten observar y definir las capacidades de una implementación de telefonía IP basada en SIP. No solo hay que tener claro el uso de herramientas como esta, sino tener muy claro las métricas utilizadas en la evaluación del protocolo SIP.

7. Trabajos Relacionados

Algunos estudios y soluciones relacionadas a proveer mecanismos de contingencia y escalabilidad en sistemas de telefonía IP existen actualmente. A lo largo de este capítulo se describirán algunos trabajos relacionados con esta investigación, donde se analizarán sus soluciones destacando sus ventajas y examinando que se puede replantear para obtener, desde el punto de vista teórico en estos momentos, mejores resultados.

7.1 High Availability for SIP: Solutions and Real-Time Measurement Performance Evaluation

En esta investigación se propone la creación de "una solución transparente y práctica de failover para servidores proxy tanto para SIP como para RTP" [26]. La razón del presente trabajo es aumentar la disponibilidad, estabilidad y escalabilidad de sistemas multimedia basados en SIP utilizando mecanismos de failover activo-pasivo con IP flotantes, tanto para la arquitectura de SIP como para la arquitectura de RTP.

Este capítulo se basará en lo especificado por el trabajo investigativo, ubicado en [26] seguido de una breve conclusión acerca de la implementación.

7.1.1 Arquitectura de Redundancia de SIP

"Para poder tener una redundancia mediante SIP es indispensable tener dos o más entidades, es necesario que todos los servidores SIP tengan conocimiento de todas las transacciones de SIP que se realicen" [26]. Esto es logrado replicando todos los mensajes recibidos por el servidor activo al backup, sin que el servidor de backup se encuentre disponible para el público. Para lograr esto, ellos proponen la creación de un demonio llamado High Availability Daemon (HAD) que se encarga de correr en cada servidor SIP incluyendo un proxy SIP.

Ahora, HAD actúa diferente dependiendo del estado que se encuentre el servidor SIP, sea activo o pasivo. Estas variaciones serán explicadas a continuación:

- **Servidor Activo**
  - La dirección IP flotante es aplicada en la interfaz primaria del servidor.
  - Al recibir una solicitud SIP, el HAD del servidor activo replica esta solicitud con todos los HAD de los servidores backup. Luego le añade la cabecera de VIA a la petición SIP y se la envía al servidor SIP local.
  - Cuando recibe una petición desde el servidor SIP local, el HAD enruta la petición según la URI de la petición.
  - Cuando el HAD recibe alguna respuesta de un paquete SIP elimina la cabecera VIA agregada anteriormente.
  - Permanentemente envía mensajes Heartbeat a los distintos backups.

- Reenvía todas las solicitudes de base de datos al cluster real de base de datos.
- Monitorea el proceso local del servidor SIP, en caso de perder comunicación con el proceso local, envía un mensaje indicando la caída de ella y deshabilita la dirección IP flotante para que al momento de levantarse algún backup a activo no se dupliquen las direcciones.

- **Servidor Backup**
  - Deshabilita la dirección IP flotante de la interfaz principal.
  - Deshabilita la interfaz de acceso a la base de datos simulando así que la base de datos está caída al servidor SIP.
  - Al recibir una solicitud replicada por el servidor activo le añade el su propia cabecera VIA y la reenvía al proceso SIP local.
  - En caso de recibir una solicitud o respuesta a alguna solicitud desde el proceso SIP local lo desecha.
  - En caso de recepción de algún mensaje Heartbeat se reinicia el temporizador, si éste se encontraba en 0, cambia su rol de backup a activo.

Hay unos cuantos puntos importantes a tomar en cuenta en este trabajo:

- El servidor SIP, tanto activo como de backup, sólo escucharán por su interfaz de loopback.
- HAD como proxy para recibir las peticiones de la red.
- En caso de que el HAD activo falle, se agregó un proceso que se encarga solo de destruir el proceso de SIP local en caso de que el HAD activo falle.

La Figura 7.1 tomada de [26] muestra la topología a nivel de SIP.

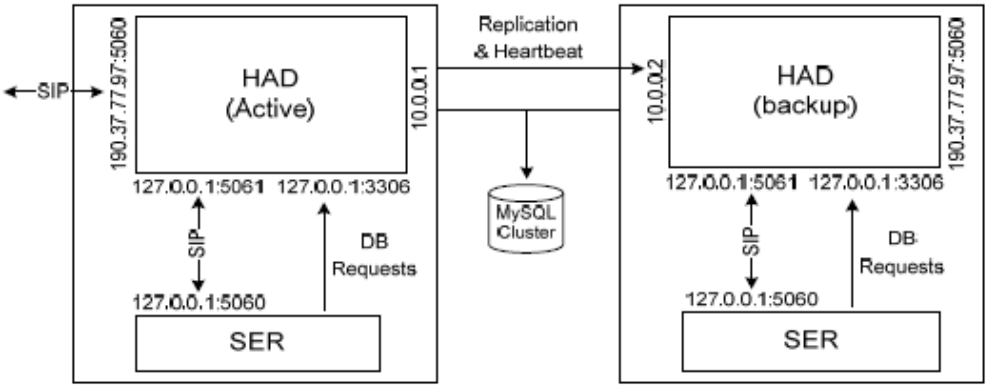


Figura 7.1: Propuesta de Arquitectura de SIP

Los servidores SIP están configurados para actuar como SIP Registrar y como SIP Proxy. Por ende, ellos tienen la capacidad de mantener el registro de las extensiones en caso de producirse el failover, y también de mantener el curso de un comienzo de sesión de SIP. Esto significa que al recibir un INVITE y ocurrir un failover mientras esta sesión sigue intentando comenzar, en el momento de recibir la respuesta, el servidor que estaba de backup podrá comenzar la sesión correctamente. Esta explicación se

puede ver claramente en la Figura 6.2 tomada de [26]:

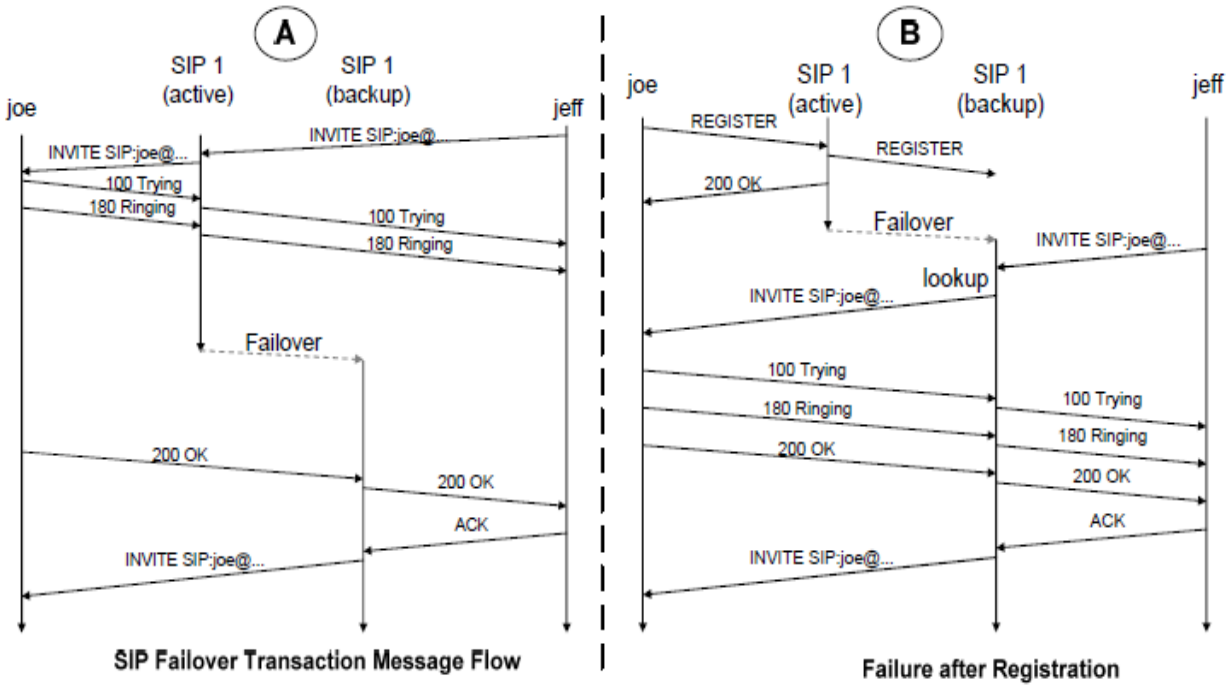


Figura 7.2: Flujo de Mensajes en Caso de Failover en Proceso de un INVITE

7.1.2 Arquitectura de Redundancia de RTP

El planteamiento de la arquitectura de RTP es muy parecida a la de SIP. Sin embargo, el proxy RTP asigna dos números de puerto por cada reenvío. En uno recibe mensajes RTP y por el otro responde. Este conocimiento previo por parte de los proxy RTP está relacionado con una lista de correspondencias que se comparte mediante HAD.

Una vez comenzada una sesión SIP para empezar el intercambio de mensajes RTP, "…el proxy RTP abre un puerto y se interconecta con el par. A este punto, el mismo no sabe cuál es la dirección IP del par ni los puertos y lo aprende al recibir el primer mensaje RTP del par, en ese momento la lista de mapeo es actualizada" [26]. Esta lista de mapeo debe ser intercambiada con otro proxy RTP. De esto se encarga HAD, así almacenando la lista de mapeo y replicándola con los otros HAD de los otros servidores RTP.

El HAD recibe las peticiones de mapeo del proxy SIP y las reenvía al proxy RTP local, además intercepta las respuestas del proxy RTP local y reenvía el puerto elegido al sistema de backup. La próxima imagen refleja la arquitectura de RTP.

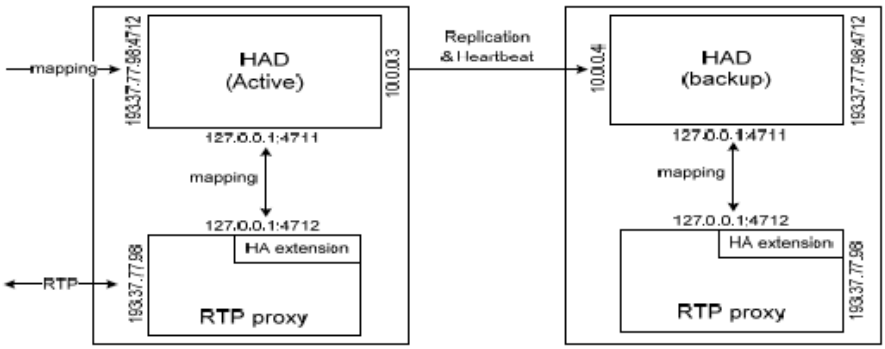


Figura 7.3: Propuesta de Arquitectura de RTP

El proxy RTP está equipado con una extensión de alta disponibilidad (HA), el cual cumple con las siguientes tareas:

- Informar al HAD de la finalización del mapeo al recibir una petición RTP.
- Especificar al par por cual puerto en particular recibirá el proxy RTP.
- Informar al HAD en caso que se requiera eliminar una entrada de la lista de mapeo, esto ocurre usualmente cuando la sesión IP es culminada.

Con este esquema, en caso de tener que aplicar failover, ambos proxies RTP tendrán la lista de mapeo sincronizada para poder mantener la comunicación por RTP activa.

7.1.3 Esquema de Balanceo de Carga

En este trabajo se empleó un mecanismo de balanceo de carga por DNS basado en registros SRV, como el explicado anteriormente en las técnicas de balanceo de carga.

Detalles de la Implementación del Balanceador de Carga

El módulo de balanceo de carga se encarga de solicitar en espacios de tiempo estáticos el número de servidores SIP disponibles en el servidor de DNS (registros DNS) y resuelve sus FQDN's (Fully Qualified Domain Name). Esto significa que por cada dominio SIP el servidor DNS tiene múltiples registros DNS para los proxies DNS agregado por ella.

El balanceador se encarga de consultar al DNS y mantener los registros de DNS. El cliente SIP al realizar una solicitud SIP llegará primero al balanceador y, basándose en la información proporcionada por el DNS y el peso de los distintos servicios, reenvía la solicitud al servidor SIP con más prioridad. Esto, sin embargo, solamente es requerido para ciertos mensajes SIP. Un cliente SIP una vez establecida la conexión (enviando un INVITE y recibiendo un OK por parte del servidor SIP), no necesita pasar de vuelta por el balanceador. Esto es logrado de dos maneras:

- El cliente maneja las mismas consultas al servidor DNS, para tener los mismos registros DNS y, al ser establecida la conexión, el cliente pueda acceder directamente al servidor SIP con quien estableció la conexión.
- Agregar las direcciones IP de los proxy SIP y el balanceador de carga. Así, la petición SIP (REGISTER, INVITE, SUBSCRIBE y OPTIONS) es gestionada por el balanceador y los otros mensajes de esa misma sesión son gestionadas

directamente por el proxy SIP.

Este trabajo se destaca por presentar una solución de alta disponibilidad utilizando esquemas de failover activo-pasivo tanto para servidores SIP como para la transmisión de mensajes multimedia mediante proxies RTP. Esta solución permite mantener las sesiones SIP establecidas o, en negociación de establecerlas, de manera funcional al aplicarse un failover.

Una gran ventaja de este trabajo es la capacidad de ofrecer un mantenimiento de la comunicación al aplicarse el failover en distintos flujos de comunicación, tanto SIP como RTP. Sin embargo, la utilización de mecanismo de balanceo de carga mediante DNS no parece ser solución eficiente de balanceo de carga. El próximo trabajo a desarrollar no implementa ningún componente nuevo e intenta utilizar herramientas existentes para lograr una solución de alta disponibilidad escalable.

7.2 On The Reliability of Voice Over IP (VoIP) Telephony

Es importante remarcar la diferencia que tiene este trabajo con el realizado anteriormente. En éste no se propuso ninguna implementación a nivel de desarrollo, sino que se utilizaron múltiples herramientas que permiten poner en práctica soluciones de clustering robustas antes desarrolladas, adaptándolas y configurándolas a una infraestructura de VoIP para aumentar y tratar de alcanzar el 99,999% de disponibilidad, conocido como THE NINES CRITERIA (Los 5 Nueves), meta que ha sido alcanzada por redes de telefonía convencionales de PSTN.

Este capítulo se basará en lo especificado por el trabajo investigativo, ubicado en [27] seguido de una breve conclusión acerca de la implementación

La configuración consiste en:

- Dos o más servidores virtuales con Kamailio.
- Dos o más servidores virtuales con FreeSwitch.

“Las llamadas son enrutadas desde un cliente hasta los servidores Kamailio usando un sistema de DNS transformando con el estándar ENUM (Electronic Number Mapping System) basado en DNS…” [27].

En caso de que un servidor Kamailio se encuentre caído, la llamada será enrutada nuevamente a otro servidor Kamailio. En caso de no encontrarse activo ninguno de esos servidores, la llamada será enrutada directamente a alguno de los FreeSwitch que se encargan de la parte de mensajería de voz, llamadas en espera y atención automática.

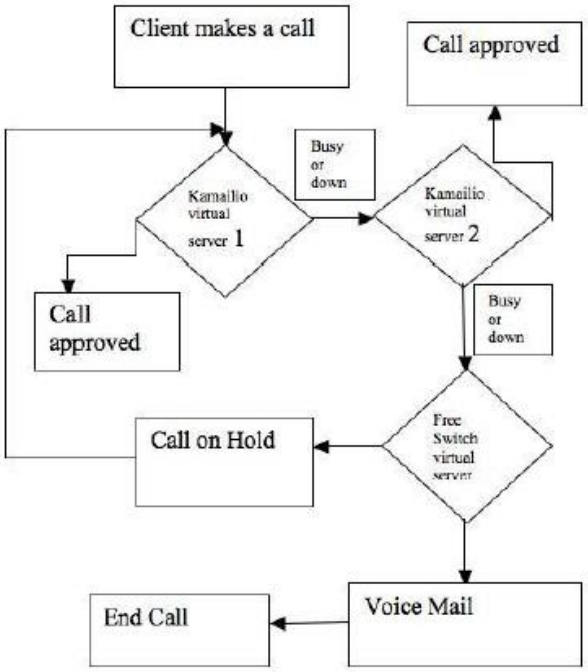


Figura 7.4: Diagrama de Flujo del Esquema Propuesto

Se puede ver en el diagrama de flujo anterior mostrado en la Figura 7.4 tomada de [27] lo explicado en el párrafo previo. El servidor Kamailio se encargará de las transacciones SIP. En caso de encontrarse caído, será atendido por otro de los servidores Kamailio disponibles y así sucesivamente. Si ningún servidor de estos se encuentra activo, los servidores FreeSWITCH se encargarán del resto de las funcionalidades.

“Para dar redundancia de hardware se propone utilizar Ultra Monkey que utiliza Linux Virtual Server (LVS) para la creación de alta disponibilidad en servicios de red” [27].

Ultra Monkey es un balanceador de carga que utiliza el protocolo Heartbeat, el cual sirve para monitorear, entre dos servidores, si se encuentran activos o no. En esta arquitectura propuesta, hay dos balanceadores con Ultra Monkey donde arman un esquema clustering activo-pasivo usando Heartbeat y en caso de que el activo deje de responder, el balanceador que se encontraba en pasivo pasará a activo y recibirá las peticiones. Heartbeat utiliza un plugin llamado IPFail que ayuda a determinar, a nivel de capa 3 (mediante mensajes ICMP), si los balanceadores se encuentran funcionando.

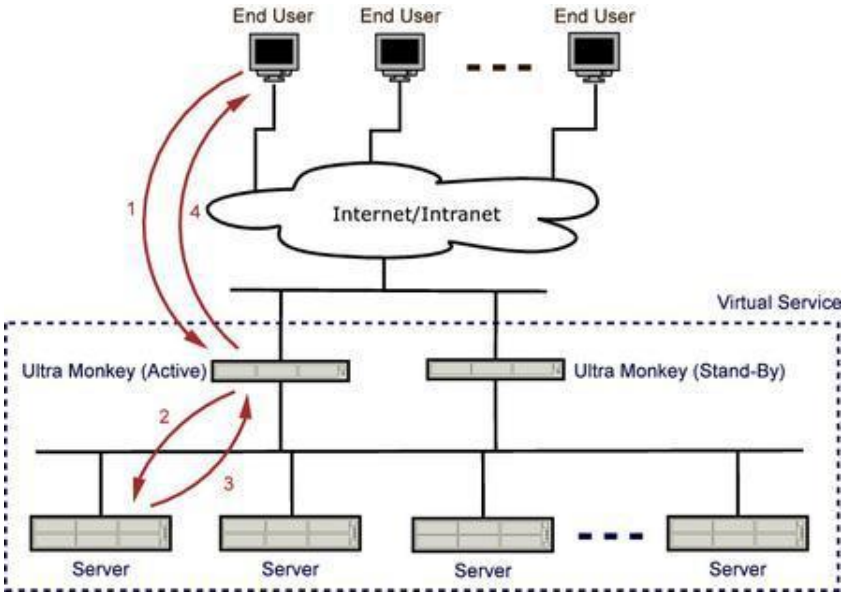


Figura 7.5: Funcionamiento de Ultra Monkey

La Figura 7.5 anterior tomada de [27] muestra el flujo por el cual los mensajes pasan por Ultra Monkey. Primero, recibe mensajes desde los clientes de VoIP, luego Ultra Monkey identifica, usualmente por capa 4 (aunque el trabajo no lo especifica), cual servidor se encuentra recibiendo mensajes por el puerto especificado en la arquitectura y se lo envía a ese servidor. Luego, las respuestas por parte del servidor al cliente pasan por el balanceador para después salir al exterior hacia el cliente.

"Ultra Monkey usa Heartbeat para manejar las direcciones IP en los hosts en los que Linux Virtual Servers corre, también monitorea el último destino de la conexión hecha a un servicio virtualizado usando el recurso IPaddr2" [27].

7.3 Design and Implementation of a System to Interconnect VoIP Services and CERN's Telephony Network

Las dimensiones de este trabajo son amplias y se orienta a proveer una interfaz de entrada a las redes de telefonía de la organización CERN (Conseil Européen pour la Recherche Nucléaire, European Organization for Nuclear Research), específicamente servicios basados en SIP. El sistema implementado sirve como punto de entrada para las llamadas originadas fuera de la red de CERN, permitiendo a usuarios que utilizan este servicio poder comunicarse con la red de telefonía de CERN (teléfonos fijos o móviles).

Debido a lo extenso del trabajo, es importante remarcar algunos puntos principales de la arquitectura implementada, empezando con la topología lógica y los componentes principales que forman parte en el trabajo.

7.3.1 Topología Lógica y Componentes

Lógicamente, el sistema consiste de múltiples componentes basándose principalmente en los tipos de servidores SIP existentes. Entre ellos, están el servidor de media (Media Server) y un SIP proxy (Proxy Server). El Media Server se encarga de proveer las funcionalidades a los usuarios SIP. Debido a que el Media Server es el que suministra la mayoría de tales funcionalidades, tendiendo, además, a utilizar muchos recursos de hardware (ya que procesan la información multimedia por software), es utilizado un Proxy server para protegerlo de la parte de señalización [28].

Por requerimientos de alta disponibilidad, la topología consta de 2 Proxies Servers y de 2 Media Servers (activo-pasivo y activo-activo respectivamente), así previniendo puntos de fallas únicos en la topología. La Figura 7.6 tomada de [28] muestra la topología lógica de la implementación planteada.

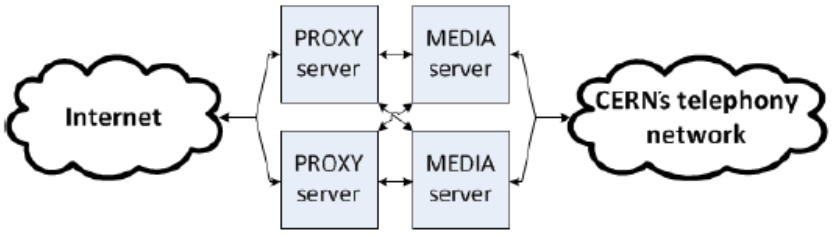


Figura 7.6: Topología Lógica

Los componentes importantes que forman parte de la topología son:

- Kamailio, como Proxy Server.
- FreeSWITCH, como Media Server.
- Corosync, como gestor de la comunicación entre los clusters.
- Pacemaker, como gestor de los recursos o servicios que forman parte en los clusters.

La topología consta de mecanismos de balanceo de carga servidos por Kamailio, incluyendo mecanismos de health checks o keepalive hacia los distintos FreeSWITCH con capacidad de failover, además de técnicas de clustering activo-activo mediante Pacemaker y Corosync entre los Media Servers. También, consta de técnicas de clustering activo-pasivo entre los Proxies Servers.

Cada Media Server está configurado para gestionar cantidades limitadas de canales mientras interrumpe las otras si el límite se encuentra ya establecido. El mecanismo de balanceo de carga permite distribuir las llamadas entre los distintos Media Servers aumentando así la escalabilidad y el límite de llamadas [28].

7.3.2 Flujos de Llamadas Deseado

Los flujos de llamadas deseados en este trabajo fueron divididos de la siguiente manera:

- Diagrama de flujo en caso de llamada valida. Esta puede verse en la Figura 7.7 tomada de [28].

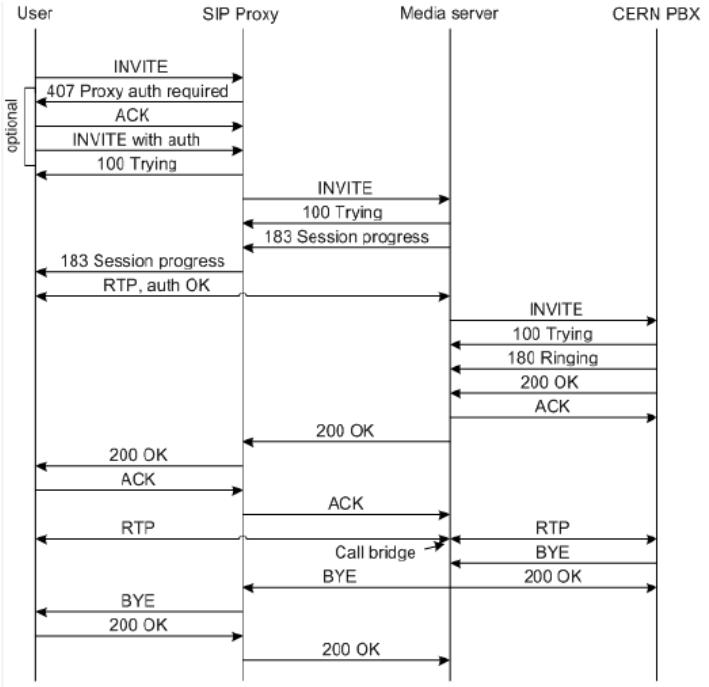


Figura 7.7: Diagrama de Flujo Para Llamada Activa

- Diagrama de flujo en caso de falla de autenticación. Esta puede verse en la Figura 7.8 tomada de [28].

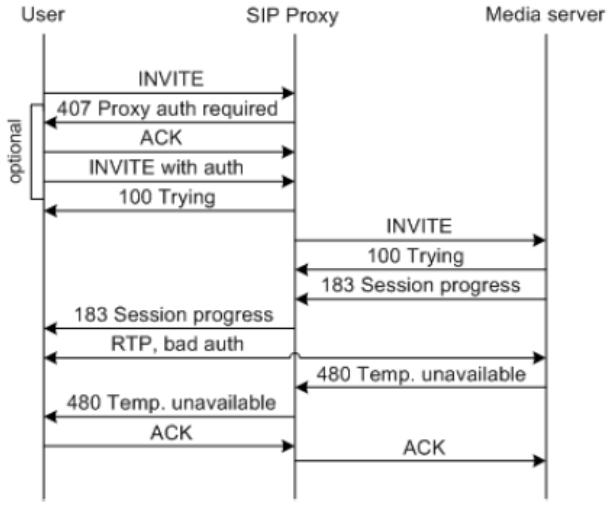


Figura 7.8: Diagrama de Flujo En Caso de Falla de Autenticación

- Diagrama de flujo en caso de llamada no autorizada. Esta puede verse en la Figura 7.9 tomada de [28].

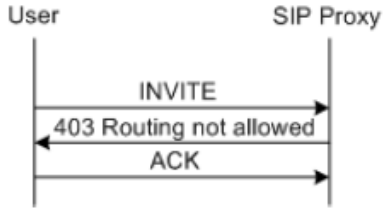


Figura 7.9: Diagrama de Flujo En Caso de Falla de Autorización

7.3.3 Funcionalidades de los Componentes Inmersos

Para simplificar la explicación de la solución implementada en este trabajo, se definirán cada uno de las funcionalidades que fueron configuradas en los componentes principales inmersos en la solución, sin detallar completamente sobre cómo fue realizada cada uno de sus funcionalidades.

Proxy Server

El Proxy Server debe aceptar conexiones con destino a protocolos de transporte tanto de TCP como de UDP. La configuración del protocolo de encriptación TLS, mediante el módulo de TLS que provee Kamailio, es utilizada y especificada con la versión TLSv1.0, según lo establecido por defecto al momento de instalación. Todo ello es así sólo cuando el protocolo de transporte utilizado es TCP.

El Proxy Server debe aceptar exclusivamente llamadas que tienen como destino a la PBX de CERN; no debe actuar como Proxy Server entre usuarios SIP [28]. El proceso de establecimiento de llamadas se mantiene al tanto del progreso de la llamada una vez respondida. Esto se logra guardando la información del dialog utilizando el módulo de Dialog provisto por Kamailio en una base de datos [28].

En relación con los mecanismos de balanceo de carga anteriormente mencionados, ellos fueron configurados utilizando el módulo Dispatcher que provee Kamailio. En esa configuración se destaca la posibilidad de ofrecer mecanismos de failover sin tener que contar con el cliente nuevamente, manteniendo previamente el estado de las transacciones por parte del proxy. También la posibilidad de ofrecer mecanismos de health checks, definiendo los intervalos en que se harán estas pruebas de health check a nivel de SIP, las cuales que permiten monitorear el estado de los Media Servers que se encuentran siendo balanceados. En caso de no recibir respuesta por parte de los Media Servers se puede considerar inactivo o caído.

Algunos mecanismos de seguridad son tomados en cuenta antes de pasar el tráfico SIP desde el Proxy Server al Media Server. Uno de los más importantes es mediante el empleo del módulo Permission que provee Kamailio. Este es utilizado, incluso, para prevenir que ciertos servicios dentro de la red de telefonía de CERN sean accedidos (servicios de conferencia por ejemplo). El módulo Permission es utilizado ampliamente en este trabajo; esencialmente, el mecanismo fue empleado controlándolo mediante el

esquema de numeración planteado e incluyendo en las reglas aquellos usuarios que no pudiesen llamar o ser llamados [28].

Además del uso del módulo Permission como mecanismo de seguridad particularmente referido al acceso, se utilizaron mecanismos para la protección de ataques, especialmente para evitar ataques de DoS (flooding) basándose en la dirección IP fuente. Este mecanismo fue implementado usando 2 módulos, pike y htable, donde pike verifica, al recibir un mensaje, cuantos mensajes han sido recibidos por el Proxy Server desde esa IP fuente (esa IP es agregada a la tabla de hash definida por htable luego de recibir un total de 16 mensajes SIP en intervalos de 2 segundos) y htable verifica si esa IP fuente se encuentra en la tabla hash (baneada); si se encuentra baneada el tráfico es descartado (según la configuración, las IPs ubicadas en la tabla hash son baneadas por un total de 5 minutos) [28].

Otro mecanismo de antiflooding configurado es el caso de intento de “password cracking” que también es gestionado por el módulo htable. En este caso se verifica si un usuario ha intentado autenticarse, con resultado fallido, más de 3 veces. Si este límite ha sido excedido, es baneado por un total de 15 minutos.

Media Server

El Media Server debe reservar un canal por cada llamada entrante y luego se procede a realizar una autenticación del usuario de la llamada de entrada. La manera en que autentica a los usuarios no es usando el módulo Mod_directory, convirtiéndolo en Servidor Registrar, sino configurando una lista de acceso donde únicamente acepte llamadas entrantes originadas por el Proxy Server y, además, un mecanismo personalizado que será explicado un poco más adelante. Los perfiles SIP fueron totalmente eliminados exceptuando el perfil Internal en el que las llamadas de entrada desde Kamailio fueron asignadas a este perfil.

Además, se utiliza un método para realizar una capa extra de autenticación de los usuarios. Esta funcionalidad varía según como el flujo de llamada es realizado, lo cual se puede ver en la Figura 6.7. Este mecanismo es creado mediante scripts escritos en Lua donde un PIN es ofrecido al usuario que se encuentra llamando mediante un IVR. Este debe colocar dicho PIN correctamente, mediante tonos DTMF, para que la autenticación sea completa.

Una vez autenticado el usuario mediante la verificación de la lista de acceso, que el mecanismo de autenticación por PIN sea correcto y que cumpla con la transmisión encriptada de RTP mediante ZRTP, se crea otra llamada (un segundo call leg, por la naturaleza de FreeSWITCH de ser B2BUA) hacia la PBX de la red de CERN. Al momento que se desee culminar la llamada, ambos call legs deben ser destruidos.

Softwares de Clustering

Debido a que lógicamente la topología no es igual a la topología física, la cual puede verse en la Figura 6.10 tomada de [28], algunas consideraciones son tomadas en la creación del clustering.

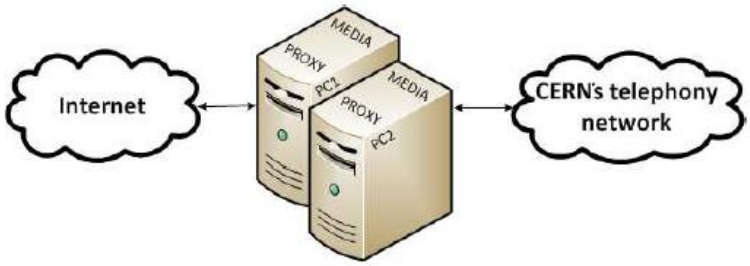


Figura 7.10: Topología Física

Por ello se crearon 4 recursos: uno para Kamailio (mediante un agente de recurso OCF desarrollado), otro para FreeSWITCH (mediante un agente de recurso OCF desarrollado), uno más para la gestión de una IP flotante (mediante un agente de recurso OCF llamado IPaddr2 que trae por defecto la instalación de Pacemaker) y por último un recurso utilizado para monitorear el cluster enviando traps SNMP (mediante un agente de recurso OCF llamado ClusterMon que trae por defecto la instalación de Pacemaker).

Debido a que la formación del cluster consta de 2 nodos y puesto que existen múltiples servicios independientes importantes corriendo, funcionalidades de STONITH y quorum son innecesarios en estos escenarios de clustering. También es importante destacar, como fue mencionado anteriormente, que el cluster de los Proxies Server están siendo gestionados como activo-pasivo, por ende múltiples recursos primitivos (IPaddr2 y Kamailio), son requeridos para la utilización correcta del servicio. Esto conlleva a la agrupación de los recursos y, si es requerida, su migración, siendo necesario mover todos los recursos agrupados.

En este trabajo es interesante destacar la capacidad de herramientas como Kamailio para poder gestionar el trafico SIP basándose en las capacidades que pueda requerir una organización como CERN. Múltiples funcionalidades fueron configuradas entre estas herramientas de software libre, las cuales permiten crear una arquitectura escalable y de alta disponibilidad con mecanismos de contingencia. Algunas de estas funcionalidades podrián ser consideradas a ser empleados en dicha propuesta.

A lo largo de este capítulo, se desarrollaron tres trabajos que implementan soluciones de alta disponibilidad escalables de servicios SIP. A partir de la propuesta que se planteará en el próximo capítulo se definirá una arquitectura que pretende proponer mecanismos de alta disponibilidad y contingencia para asegurar la misma. Además, se proponen mecanismos que ofrecen un aumento de la escalabilidad en ambientes donde el servicio puede verse afectado por altos flujos de mensajes SIP. Estos escenarios serán simulados con herramientas de estrés que permitirán generar altos flujos de mensajes SIP.

8. Marco Metodológico

En este capítulo se define la metodología que permitirá abarcar los objetivos y problemas planteados en el capítulo 2 con el objeto de implementar una arquitectura de alta disponibilidad para centrales telefónicas basadas en VoIP de manera organizada.

8.1 Adaptación de la Metodología de Desarrollo

A continuación se definen los lineamientos necesarios para realizar la implementación de una arquitectura de telefonía VoIP, los cuales serán llevados a cabo de manera ordenada para permitir alcanzar las metas esperadas. Lineamientos:

- Diseño general de la implementación.
- Implementación de ambientes virtualizados.
- Diseño de esquemas de clustering.
- Mecanismos de Alta Disponibilidad.
- Mecanismos de Balanceo de Carga.
- División de Servicios (Servicios SIP y tráfico multimedia).
- Verificación, demostración y análisis de resultados.

8.2 Diseño General de la Implementación

Es indispensable en la implementación de una arquitectura, realizar el diseño de la topología de la misma. Esto es, tanto a nivel de topología de red, ambientes en la que será implementado y los componentes inmersos que formarán parte de dicha arquitectura. El proceso de creación del diseño permite definir:

- **Arquitectura Planteada:** Esto conlleva a una observación de los componentes habilitados para el desarrollo de dicha arquitectura, lo cual permite definir estrategias y el desarrollo general de la misma basándose en los componentes disponibles para su implementación.
- **Requerimientos de la arquitectura:** De acuerdo al plante miento anterior, definir los requerimientos de la arquitectura para su diseño. Estos requerimientos pueden ser a nivel de red, hardware, ambientes de instalación e incluso requerimientos de recursos y tráfico de red para el buen funcionamiento de las distintas herramientas a utilizar en la arquitectura.

8.3 Implementación de Ambientes Virtualizados

Debido a que en el análisis de requerimientos de la arquitectura se encontraron ciertas limitaciones físicas (tanto a nivel de aprovisionamiento de red como de hardware) para la implementación de la misma, se decide la utilización de ambientes virtualizados con la utilización de un servidor, el cual será instalado con el hipervisor Xen Project. Este hipervisor permitirá la gestión de una topología de red virtualizada, en conjunto con la creación de interfaces “bridge” que proveerá el sistema operativo, para la gestión de switches virtuales y, por parte del hipervisor, la creación de distintas máquinas virtuales para la instalación y configuración de las distintas herramientas requeridas en la arquitectura.

8.4 Diseño de Esquemas de Clustering

Entre los objetivos de este trabajo se hace referencia al ofrecimiento de mecanismos de contingencia en caso de fallas dentro de la arquitectura. Dichas contingencias pueden ser logradas mediante la creación de esquemas de clustering para ofrecer mecanismos de monitoreo de servicios y alta disponibilidad de los mismos. Ahora, los esquemas de clustering, además, deberán cumplir con ciertas condiciones dependiendo del escenario y los servicios a ofrecer. Algunos de estos son:

- **Información compartida entre los nodos del cluster:** Conlleva a la necesidad de gestionar almacenamientos compartidos para los distintos componentes dentro de la arquitectura, algunos de estos podrían ser gestionados mediante la creación de iSCSI Target y también por herramientas como DRBD y la instalación y configuración de una base de datos para aquella información que pueda ser compartida por este mecanismo.
- **Infraestructura de comunicación y monitoreo de servicios:** Para la implementación de un esquema de clustering, es necesario que los componentes mantengan un flujo de monitoreo constante para verificar el estado de los mismos y los servicios que son gestionados por ellos. Tal cometido se logrará mediante la configuración e instalación de herramientas como Corosync y Pacemaker para la comunicación entre los componentes y el monitoreo y acciones a tomar, respectivamente.
- **Contingencia contra errores dentro del esquema:** Dichos esquemas de clustering pueden tener fallas en sus procesos de comunicación, como también el posible mal funcionamiento entre servicios que se encarguen de la gestión de almacenamientos compartidos. En caso de ocurrir alguno de estos fenómenos, deberán existir mecanismos de contingencia para evitar un mal comportamiento en la arquitectura o la posible corrupción de datos. Esto es logrado mediante la configuración de mecanismos de fencing.

8.5 Mecanismos de Alta Disponibilidad

Ofrecer mecanismos de alta disponibilidad es el resultado de la creación y configuración de esquemas de clustering, monitoreo de servicios y funcionalidades de algunas de las herramientas que formarán parte de la arquitectura. Principalmente, ofreciendo mecanismos de failover, que pasan desde failover a nivel de nodos, de servicios o de las mismas transacciones SIP.

8.6 Mecanismos de Balanceo de Carga

El ofrecimiento de mecanismos de balanceo de carga permitirá a la arquitectura escalar los servicios provistos de manera ascendente. Además de eso, permite a una arquitectura proveer un servicio más acorde y eficiente en escenarios de alta demanda donde la arquitectura se encuentre a prueba. Por ello, la evaluación para proveer mecanismos de balanceo de carga, junto a los de alta disponibilidad forma parte del núcleo de este trabajo para llegar a los objetivos planteados.

8.7 División de Servicios

Cuando se disponen de múltiples herramientas con la capacidad de ofrecer una misma funcionalidad, es importante tener en cuenta cuál de ellas puede proveerla de la mejor manera o tengan alguna otra consideración relevante para el mismo. A lo largo de este trabajo, se describirán las ventajas de la división de funcionalidades de ciertos servicios con las pruebas correspondientes para respaldar dichas decisiones.

8.8 Monitoreo y Gestión de Servicios y Carga

Como fue comentado en la Sección 8.4, existe un componente que se encarga del monitoreo de los servicios relevantes dentro de la arquitectura. Las razones se deben a que es importante verificar el estado funcional de los servicios dentro de la infraestructura de red para efectuar mecanismos de failover en ciertos casos de falla, o la terminación correcta del mismo, para evitar intermitencias y un mal funcionamiento. Así mismo es importante efectuar monitoreo de la carga. Dicha carga es gestionable con la configuración de algoritmos de balanceo de carga que permitirá el balanceo de la carga basándose en esta premisa.

8.9 Verificación y Análisis de Resultados

Basado en la arquitectura que se desea implementar, es importante verificar el comportamiento de algunos de los mecanismos que han sido mencionados a lo largo de este capítulo. Entre ellos están:

- Mecanismos de alta disponibilidad y failover de los distintos servicios relevantes que forman parte de la arquitectura.
- Mecanismos de contingencia en caso de fallas dentro de los esquemas de clustering.
- Verificación de los mecanismos de failover que ocurren entre los flujos de mensajes SIP, especialmente en transacciones que conllevan al establecimiento de llamadas.

En conjunto a lo mencionado anteriormente, es requerido el análisis de resultados de los siguientes aspectos:

- **Tiempos de Respuesta:** En arquitecturas de alta disponibilidad es imprescindible que los tiempos de respuesta a los distintos mecanismos de contingencia sean aceptables, este análisis depende también de los servicios y los esquemas inmersos en la arquitectura.
- **Pruebas de Estrés:** Debido a que se desea incluir mecanismos de balanceo de carga, división de servicios y aspectos de alta disponibilidad, es importante la realización de pruebas en distintos escenarios y flujos de mensajes que coloquen a prueba la arquitectura desde el punto de vista de carga. Hay consideraciones, en la realización de dichas pruebas, que salen del alcance de este trabajo debido a la limitación de los recursos de hardware disponibles, sin embargo, es importante verificar que el comportamiento sea el óptimo en situaciones de alta carga para justificar las decisiones tomadas en algunos aspectos implementados en la arquitectura.

9. Arquitectura de Alta Disponibilidad de Telefonía VoIP

9.1 Introducción

En cualquier tipo de solución de comunicación, las fallas son un aspecto que se encuentra presente permanentemente en un sistema. Por ello, se requieren mecanismos de contingencia para solventar estos fallos de manera automática, con el menor tiempo posible y, dentro de sus posibilidades, transparentes para el acceso al usuario. Otros aspectos son implementados en la arquitectura, algunos de estos son:

- Mecanismos de Alta Disponibilidad.
- Mecanismos de Balanceo de Carga.
- Solventar el fenómeno de NAT Traversal.

En este capítulo, se describe en detalle lo implementado en la arquitectura de telefonía VoIP para lograr los objetivos planteados en el Capítulo 2, siguiendo el conjunto de lineamientos que fueron descritos el capítulo anterior.

9.2 Diseño General de la Solución

En la Sección 8.1.1, se comenta acerca de la importancia de determinar las consideraciones para el diseño de la arquitectura a implementar. Debido a algunas limitaciones a nivel de recursos provistos por el ambiente en el cual la arquitectura será implementada, se logra la creación del siguiente diseño mostrado en la Figura 9.1.

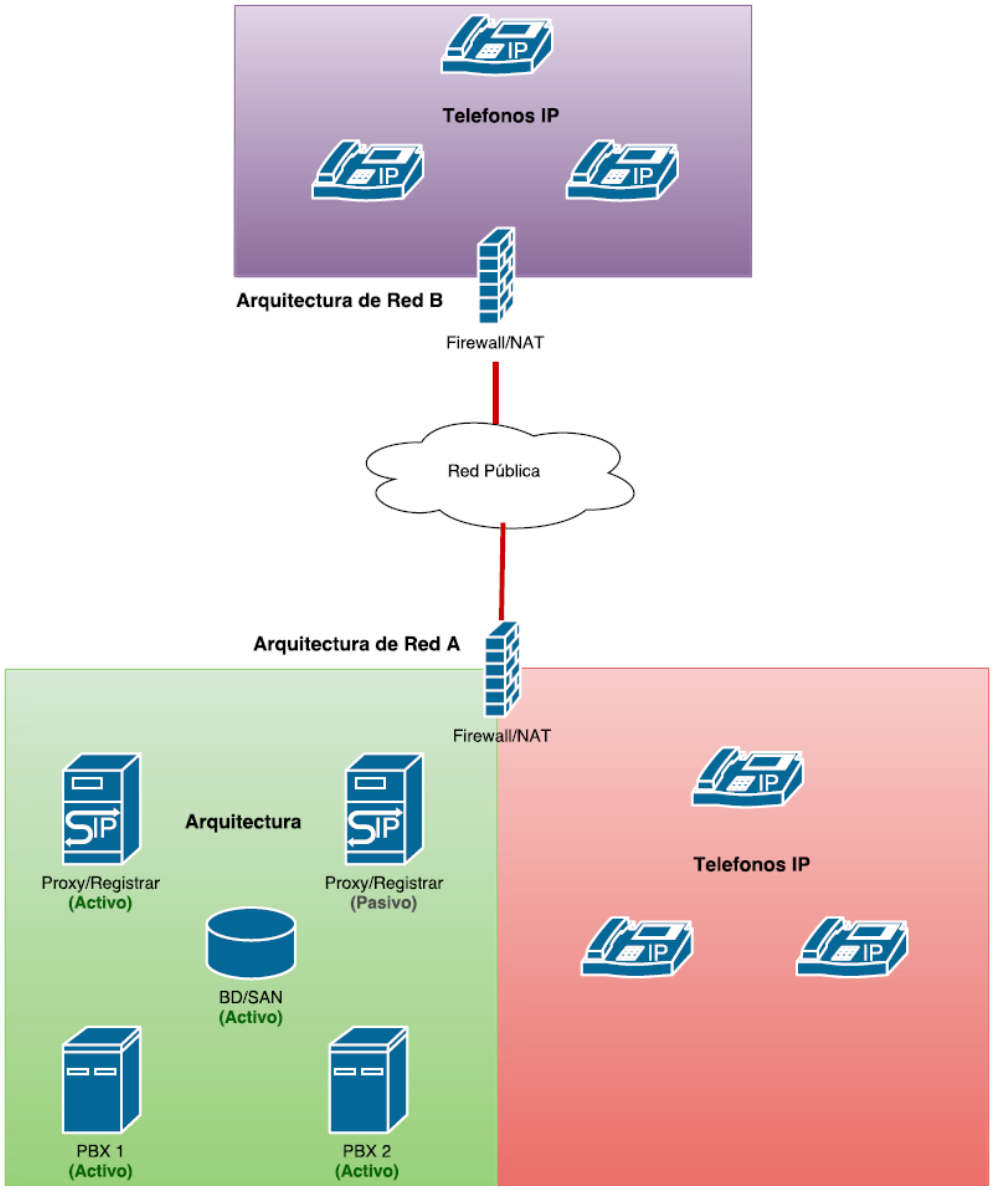


Figura 9.1: Diseño General de la Arquitectura

La Figura 9.1 refleja la arquitectura implementada. En la arquitectura de red A, del lado izquierdo (dentro del cuadro verde), se encuentra la arquitectura de alta disponibilidad de telefonía, los elementos inmersos en dicha arquitectura son los siguientes:

- **Proxy/Register:** Este es el componente principal encargado de los servicios provistos por el protocolo de señalización SIP, este se encuentra actuando como un proxy SIP con estado para permitir realizar los mecanismos de balanceo de carga hacia las entidades PBX. Además, actúa como proxy RTP para solventar problemas que se encuentran por el fenómeno de NAT Traversal.
- **PBX:** Este componente actúa como un servidor multimedia. Es utilizado como receptor secundario de tráfico SIP (en caso del requerimiento de establecimiento de llamadas) y dichos servidores se encuentran siendo balanceados por el componente que actúa como proxy SIP en la arquitectura.

- **BD/SAN:** Este componente actúa como un almacenamiento compartido para los dos componentes definidos anteriormente y mantiene la información centralizada para el acceso a dicha información de manera transparente por cualquier instancia de esos componentes.

Cada instancia de los distintos componentes, en el diseño de la Figura 9.1, indican si ella se encuentra activa, lo que significa que se encuentra proveyendo servicios, o pasiva, lo que especifica que se encuentra sin proveer servicios. Estos indicativos hacen referencia a la implementación de esquemas de clustering que son utilizados como mecanismos de contingencia dentro de la arquitectura. Cada esquema de clustering es formado por múltiples instancias de un mismo componente. Se comentará con mayor detalle acerca de los esquemas de clustering más adelante en este capítulo.

El resto de los componentes que forman parte del diseño son los siguientes:

- **Firewall/NAT:** Estos componentes actúan como delimitadores de las redes internas con las redes públicas de las distintas arquitecturas de red representadas en la Figura 9.1. Además, son utilizados para la interconexión con las distintas redes de la oficina. La finalidad de dichos componentes es principalmente la de lograr la simulación del fenómeno de NAT Traversal teniendo a usuarios SIP fuera de la arquitectura de red A donde se encuentra la arquitectura de telefonía implementada.
- **Teléfonos IP:** Estos componentes actúan, claramente, como los endpoints que desean comunicarse utilizando la arquitectura de telefonía implementada.

9.3 Diseño Especifico de la Solución

En este punto se menciona el diseño de la solución de manera más específica, en donde se resaltan, con mayor profundidad, los siguientes aspectos:

- Ambiente en el cual la arquitectura es desplegada.
- Topología de Red.
- Componentes inmersos en la arquitectura.
- Diseño de los esquemas de clustering.
- Flujos de Comunicación (tráfico SIP y RTP).

Como se verá en la Figura 9.2, se cuenta con un servidor para la realización de la implementación. Las especificaciones de dicho servidor se pueden ver en la Tabla 9.1.

Componente	Descripción
Sistema	ThinkServer TS140 Lenovo
Procesador	Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz. 4 Nucleos
Memoria	16 GB, 4 slots con 4 tarjetas de 4 GBs cada una
Disco Duro	1 TB Western Digital WD10EZEX-00B • 1 GB Partición /boot • 80 GB Partición RAID Autodetect (Ubicación de Dominio 0) • 850 GB Partición RAID Autodetect 1 TB Western Digital WD10EZEX-08M • 1 GB Partición /boot • 80 GB Partición RAID Autodetect (Ubicación de Dominio 0) • 850 GB Partición RAID Autodetect
Interfaz de Red	eth0: Ethernet Connection I217-LM 1Gbit/s

Tabla 9.1: Especificaciones del Servidor Huesped de la Arquitectura

La Figura 9.2 muestra el diseño de la solución de manera más profunda en aspectos de topología de red y el ambiente en el que fue desplegado:

Interfaz Bridge	Interfaz	Subredes Generadas	Subredes (IP)
xenbr0	eth0 (interfaz física)	**Acceso Internet**	**192.168.7.0/24**
xenbr1	lo:1	**Red VoIP**	**10.0.1.0/24**
xenbr2	lo:2	**Red de Datos**	**10.0.2.0/24**
xenbr3	lo:3	**Red de Monitoreo**	**10.0.3.0/24**
xenbr5	lo:5	**Red Pública**	**10.0.5.0/24**
xenbr6	lo:6	**Arquitectura de Red B**	**10.0.6.0/24**

Tabla 9.2: Relación entre Interfaces y Subredes Virtualizadas

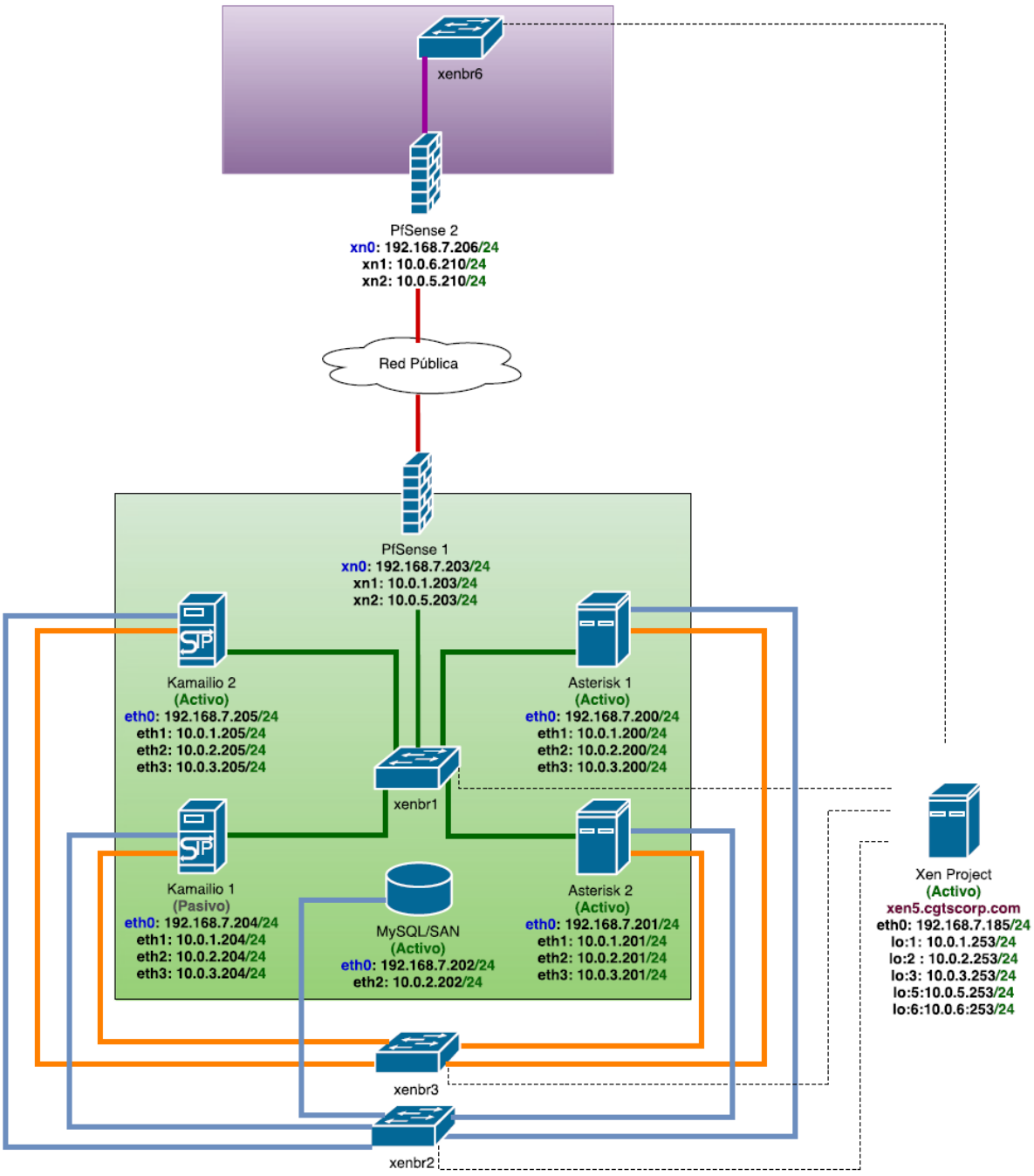


Figura 9.2: Diseño Específico de la Arquitectura

9.3.1 Implementación de Ambiente Virtualizado

Como fue comentado en la Sección 8.1.2, hay limitaciones físicas para el aprovisionamiento de un diseño de este estilo en un ambiente físico, es por ello que se opta por realizar el diseño anteriormente mostrado en un ambiente virtualizado. Para lograr esto, se cuenta con un servidor, con las especificaciones descritas en la Tabla 9.1, donde es montado un hipervisor llamado **Xen Project v4.6** que, como muestra la Figura 9.2, es el encargado principal del gestionamiento de la arquitectura implementada.

Xen Project es un hipervisor tipo 1 que permite la virtualización de distintas instancias de máquinas de manera paralela en una misma máquina (en este caso, el servidor provisto por la empresa). Para lograr esta autonomía de distintas instancias corriendo distintos sistemas operativos gestionados por una misma máquina, Xen divide la gestión de su sistema en distintos dominios:

- **Dominio 0:** el dominio de control principal y el que se encarga de la gestión del resto de los dominios y las máquinas virtuales creadas en estos.
- **Otros Dominios:** En estos dominios es donde se encuentran las máquinas virtuales siendo gestionadas por los mismos. Dichos dominios tienen restricciones de acceso y conocimientos a la información del sistema principal y acceso a los controladores del sistema. Todas las máquinas de la arquitectura se encuentran gestionadas por el Dominio U.

Para la creación de la topología de red virtualizada y el acceso a Internet, se utiliza el método de **bridging**. Este método solo consiste en la capacidad del sistema operativo encontrado en el Dominio 0 (o dominio de control) para la creación de interfaces "bridge" (o interfaces puentes) contra interfaces físicas (para el acceso a Internet) o loopbacks (para la creación de otras subredes). Así, las distintas interfaces virtuales creadas por Xen en el Dominio 0 que se mapean con las interfaces de las máquinas virtuales en el Dominio U, pueden comunicarse entre sí mediante el switch virtual generado por la interfaz bridge creada anteriormente. La explicación de cómo Xen permite esta solución se ve en la Figura 9.3[4].

[4] https://wiki.xenproject.org/wiki/Xen_Networking

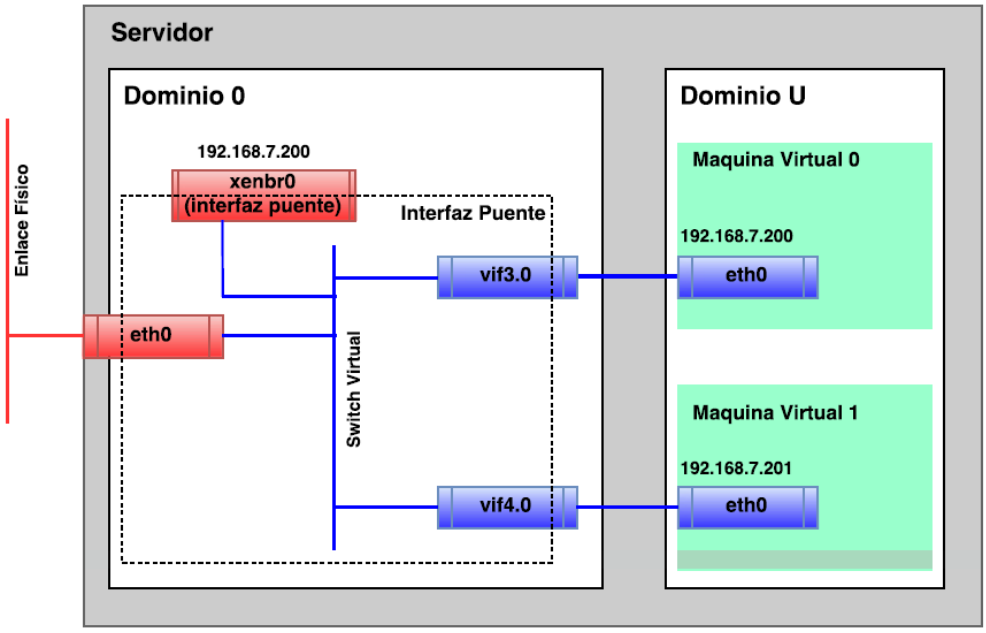


Figura 9.3: Mecanismo de Bridging con Redes Xen

Como lo especifican la Figura 9.2 y la Tabla 9.2, se crean 5 interfaces bridge (xenbrX, donde X es un número entero del 0 al 6 excluyendo el 4), 4 interfaces loopback (lo:X donde X es un numero entero del 0 al 6 excluyendo al 4) y la interfaz física eth0. Estas son mapeadas de la manera que lo indica la Tabla 9.2 para permitir la creación de las subredes de manera virtualizada y para acceder a Internet por la interfaz física eth0.

9.3.2 Topología de Red

La topología de red se encuentra en la Tabla 2.2, la misma consiste en:

- 192.168.7.0/24 (Subred de acceso).
- 10.0.1.0/24 (Subred de tráfico de telefonía VoIP).
- 10.0.2.0/24 (Subred de tráfico de data, tanto acceso a SAN como acceso gestión de Base de Datos).
- 10.0.3.0/24 (Subred de tráfico de clustering).
- 10.0.5.0/24 (Subred utilizada para la simulación de una red pública).
- 10.0.6.0/24 (Subred utilizada para la gestión de una subred en la arquitectura de red B).

Las razones por las cuales el diseño de la topología de red es realizada de esta manera, se especifican en la Sección 9.3.2.1.

9.3.2.1 División de Tráfico de Red

La división de tráfico en distintas subredes se debe a que ciertos tráficos tienen diferentes necesidades de red (ancho de banda por ejemplo) y deficiencias en caso de congestionamiento. Las necesidades de algunos de estos son:

- **Tráfico VoIP:** Todo tráfico en tiempo real (en este caso, tráfico RTP) se ve afectado drásticamente, desde el punto de vista del usuario, cuando las redes que lo gestionan se encuentran congestionadas. Desde el punto de vista del

usuario, se transforma en un delay o directamente en ruido cuando se encuentra activo en una llamada telefónica por VoIP.

- **Tráfico de Datos:** Los servicios de almacenamiento compartido como iSCSI conllevan al consumo alto de ancho de banda, lo que pueden producir alto congestionamiento. Por ello, es gestionado en una subred aparte, en conjunto con el tráfico relacionado al acceso a base de datos.
- **Tráfico de Monitoreo:** Este tráfico es sumamente sensible, debido a que la comunicación entre los nodos que forman los distintos esquemas de clustering es síncrona, y en caso de no recibir los mensajes en el momento requerido conllevan al fenómeno de Split-Brain o el mal funcionamiento de aquellos componentes que se encargan de la gestión de bloqueos y accesos a almacenamientos compartidos que podrían resultar en la corrupción de datos en caso de ocurrir dichos mal funcionamientos.

9.3.2.2 Firewalls

En cuanto a los Firewalls descritos en el diseño, estos son máquinas virtuales corriendo instancias de **PfSense v2.2.6**. La razón por la cual se escoge PfSense es por su bajo consumo y sus funcionalidades. Los Firewalls son utilizados como delimitadores de las redes internas con las redes públicas de las distintas arquitecturas de red representadas. Esto permite realizar las pruebas con usuarios detrás de otra NAT y solventar los problemas del fenómeno NAT Traversal.

Pocos cambios fueron realizados a nivel de configuración, sin embargo es importante destacar algunas funcionalidades configuradas:

- **PfSense 1:** Esta es la instancia de Firewall/NAT que se encarga de gestionar la arquitectura de red donde se encuentra la implementación de telefonía VoIP desplegada, que además da acceso a la red pública simulada, y acceso a las redes de la oficina y a Internet. Ciertos aspectos son relevantes en este componente los cuales fueron configurados así:
  - **NAT 1:1:** A diferencia de una configuración de NAT común (teóricamente conocido como PAT), en este caso se configura un mapeo de dirección pública y privada 1 a 1, también conocido como NAT estático. Esto permite casar una dirección pública con una dirección privada, todo tráfico de salida es traducido con la dirección pública y la de entrada con la dirección privada.
  - **Apertura de puertos para tráfico RTP:** Se permite el acceso al tráfico con puertos de destino igual al especificado en la regla y con dirección pública igual a la casada en la configuración de NAT 1:1.
  - **Apertura de puertos para tráfico SIP:** Se permite el acceso al tráfico con puertos de destino igual al puerto UDP bien conocido de SIP en la regla y con dirección pública igual a la casada en la configuración de NAT 1:1.

- **PfSense 2:** Esta es la instancia de Firewall/NAT que se encarga de gestionar la arquitectura de red B, en el cual se encuentran los endpoints donde se desea simular el fenómeno de NAT Traversal, que además da acceso a la red pública

simulada, y acceso a las redes de la oficina y a Internet. Ciertos aspectos son relevantes en este componente, los cuales fueron configurados así:

- **NAT:** A diferencia de PfSense 1, en este caso si se configura un NAT dinámico, así cualquier dirección dentro de esta arquitectura de red B puede salir con la dirección privada transformada por la dirección pública.
- Este Firewall no posee ninguna regla que permita el acceso a las subredes internas desde la red pública.

9.3.3 Componentes Inmersos en la Arquitectura

Como fue mencionado anteriormente, existen ciertos componentes que permitieron otorgar las funcionalidades a la arquitectura de telefonía, entre ellos están:

- **Kamailio:** Este componente (mencionado en la Sección 9.2 como Proxy/Registrar) corre las siguientes herramientas para proveer los servicios.
  - **Kamailio v4.3:** Esta es la herramienta que convierte a este componente en un proxy SIP con estado, así ofreciendo mecanismos de failover de transacciones SIP y mecanismos de balanceo de carga. Además se comportará como un Registrar Server y un Location Server.
  - **RTPProxy v2.0.0:** Esta herramienta es utilizada para convertir al componente también en un proxy RTP en caso de ser necesario (contra fenómenos de NAT Traversal).
- **Asterisk:** Este componente (mencionado en la Sección 9.2 como PBX) se encuentra corriendo la herramienta **Asterisk v13.1**. Este principalmente actúa como un servidor multimedia con la capacidad de ofrecer múltiples funcionalidades. Entre ellas están, transcoding, servicio de voicemail, IVR, transferencia de llamadas, etc.
- **MySQL/SAN:** Este componente (mencionado en la Sección 9.2 como **BD/SAN**) actúa como un almacenamiento compartido para los dos componentes definidos anteriormente. Debido a que cierta información es almacenada en base de datos y otra a nivel del sistema de archivo, existen dos herramientas que se encuentran ofreciendo estos servicios
  - **MySQL Community Edition v5.7:** MySQL es utilizado como sistema manejador de base de datos para mantener información compartida entre los distintas instancias de los distintos componentes que forman parte de la arquitectura
  - **Linux-IO Target (targetcli v2.1):** Esta herramienta es utilizada para convertir a esta instancia en un servidor SAN del cual se configura un iSCSI Target para permitir a las distintas instancias de servidores Asterisk crear sesiones iSCSI contra ese Target y poder gestionar este almacenamiento de manera compartida. Esto es necesario debido a que cierta información de Asterisk (algunos archivos de configuración, archivos multimedia como los del buzón de voz, music on hold, etc) debe ser compartida por ambas instancias para el buen funcionamiento de la arquitectura.

9.3.4 Diseño de los Esquemas de Clustering

En este trabajo existen dos esquemas de clustering de dos nodos cada uno, uno para las distintas instancias de los componentes que corren Kamailio y RTPProxy y otro para las distintas instancias de los componentes que corren Asterisk. Las razones detrás de estos esquemas son, como fue explicado anteriormente, ofrecer mecanismos de contingencia en caso de errores, gestionar almacenamientos compartidos y monitorear el estado de los distintos servicios relevantes que forman parte de los distintos esquemas. Estos esquemas de clustering son formados mediante el gestor de recurso **PaceMaker v1.1.13** y la capa de comunicación y creadora de la infraestructura de cluster **Corosync v2.3.4**.

9.3.4.1 Esquema de Clustering de Componentes Proxy/Registrar

Este es un esquema de clustering Activo/Pasivo, lo que significa que una de las instancias es la que se encargará de tener activo todos los servicios y se encargará de proveerlas, mientras la otra instancia o nodo se encuentra a la expectativa en caso de que algún error pueda presentarse en la instancia o nodo activo. Dicho esquema queda de la siguiente manera:

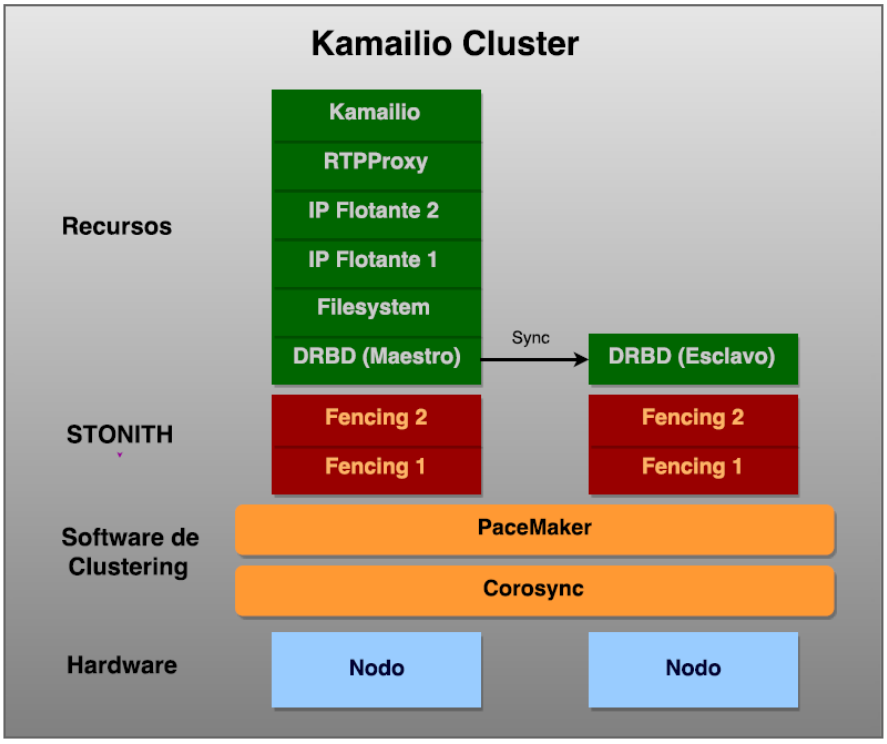


Figura 9.4: Esquema de Clustering de Componentes Proxy/Registrar

En la Figura 9.4 se puede notar que los distintos recursos (son llamados así a los servicios gestionados por algún agente de recurso por parte de PaceMaker) se encuentran montados y activos en un único nodo. Este actúa como el nodo activo que proveerá los servicios excepto uno de ellos, dicho recurso es un recurso clonado con multi-estados, ya explicado anteriormente.

La utilización de direcciones IP flotantes (floating IP), en estos casos, son relevantes para la creación del socket de Kamailio con esas direcciones especificadas. Así existirá la transparencia por parte del usuario en la manera que accede al servicio al momento de ocurrir alguna falla.

Los dispositivos o recursos STONITH especificados en la Figura 9.4 son aquellos agentes que se encargarán de aplicar los mecanismos de contingencia en caso de ciertos errores dentro del esquema de clustering, los cuales son resueltos aplicando fencing sobre el nodo con problemas. Las razones de este esquema radican en la necesidad de mitigación de la posible existencia de fenómenos como el de Split-Brain o fallas en el recurso de DRBD que conllevaría a la corrupción de datos.

Para la configuración de mecanismos de fencing, se suelen tener dispositivos de fencing (hardware) destinados al monitoreo y ejecución de medidas de contingencia. Debido a la ausencia de dichos dispositivos de fencing, pero contando con que la implementación es montada en un entorno virtualizado utilizando un hipervisor como Xen, existen módulos que permiten que estos mecanismos de fencing sean monitoreados y realizados por el mismo hipervisor. Cuando se abarque la configuración de estos mecanismos se explicará en detalle cómo se realiza.

Ahora, la acción tomada por el cluster en caso de percibir alguna falla en alguno de los recursos dentro del cluster o el nodo activo, es la de migrar dichos recursos hacia el nodo pasivo y modificar el estado del recurso DRBD de Esclavo a Maestro en el nodo pasivo. Esto hará que el nodo pasivo ahora sea el que se encargue de proveer el servicio y convertirse en nodo activo.

9.3.4.2 Esquema de Clustering de Componentes PBX

Este es un esquema de clustering Activo/Activo, lo que significa en este caso, que ambas instancias se encargan de tener activos todos los servicios y proveerlos. Este esquema tiene la funcionalidad primaria de prevenir la corrupción de datos del almacenamiento compartido entre ambas instancias y de monitorear a mayor granularidad los componentes que se encargan de prevenir la corrupción de dichos datos y el estado del servicio de Asterisk para, en caso de fallas, terminar el servicio en el nodo donde la falla ha ocurrido evitando mal funcionamiento de la arquitectura. Dicho esquema queda como mostrado en la Figura 9.5.

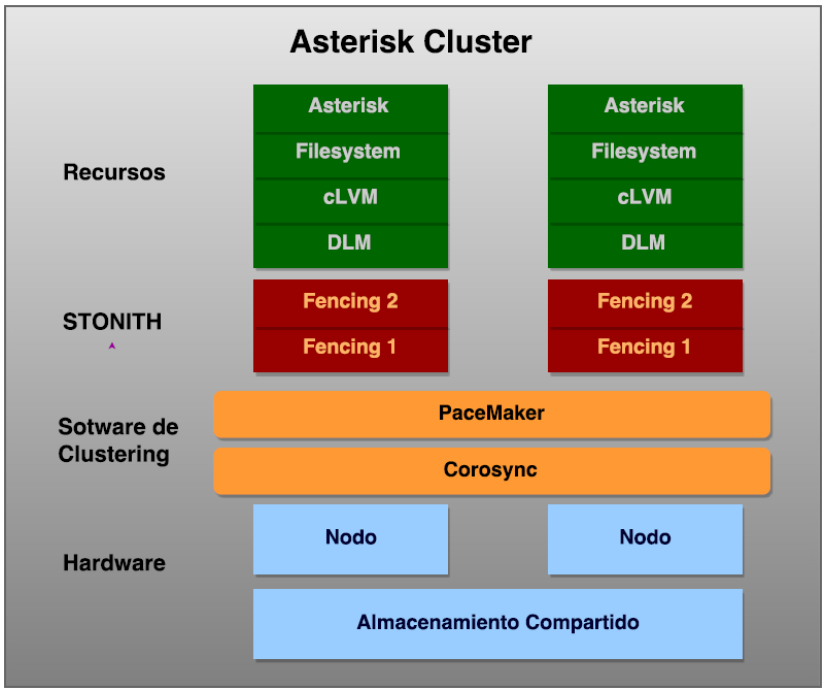


Figura 9.5: Esquema de Clustering de Componentes Proxy/Registrar

Más adelante, en este trabajo se especificará la configuración y parámetros tomados en cuenta en cada uno de los esquemas de clustering creados.

9.3.5 Flujo de Comunicación

En este punto, se describe el flujo de mensajes por el cual, en los distintos ambientes, los endpoints y la arquitectura se ven sometidos en los casos principales de tráfico de VoIP, como es el establecimiento de llamadas entre otro endpoint y el registro de un endpoint a la arquitectura.

9.3.5.1 Flujo de Comunicación en el Establecimiento de una Llamada

En caso del establecimiento de una llamada, el flujo se basa en la creación de una sesión INVITE con la utilización de un proxy SIP con estado en el medio de la comunicación. En este flujo, debido a que se cuenta con Asterisk en la comunicación, el cual es un Back-to-Back User Agent (B2BUA), el mismo crea otra sesión INVITE para el establecimiento de la comunicación con el endpoint. Estas sesiones creadas son resaltadas en el flujo con un Call-ID diferente. Además, Kamailio es el encargado de autenticar al endpoint SIP que desea establecer la llamada, aliviando de esta funcionalidad a Asterisk. El flujo de comunicación es destacado en la Figura 9.6.

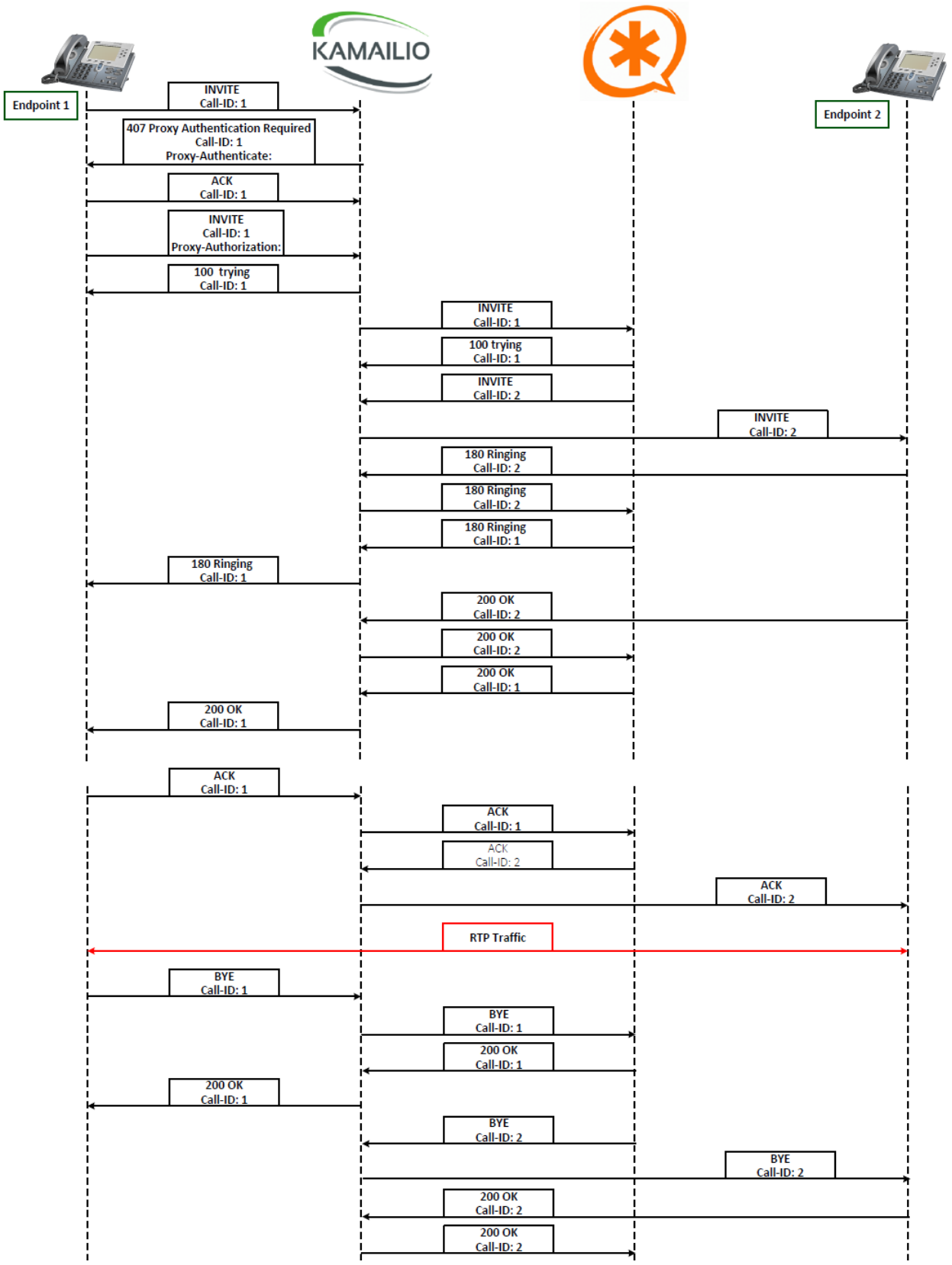


Figura 9.6: Flujo de Comunicación en el Establecimiento de una Llamada

9.3.5.2 Flujo de Comunicación en el Registro de un Endpoint

En el caso de que un endpoint requiera registrarse contra la arquitectura, el flujo se establece únicamente con Kamailio debido a que actúa también como un Registrar Server. El flujo sigue el intercambio de mensajes según lo establecido en el RFC 3261, incluyendo el proceso de autenticación esperado para la validación del usuario. El flujo de comunicación es descrito en la Figura 9.7:

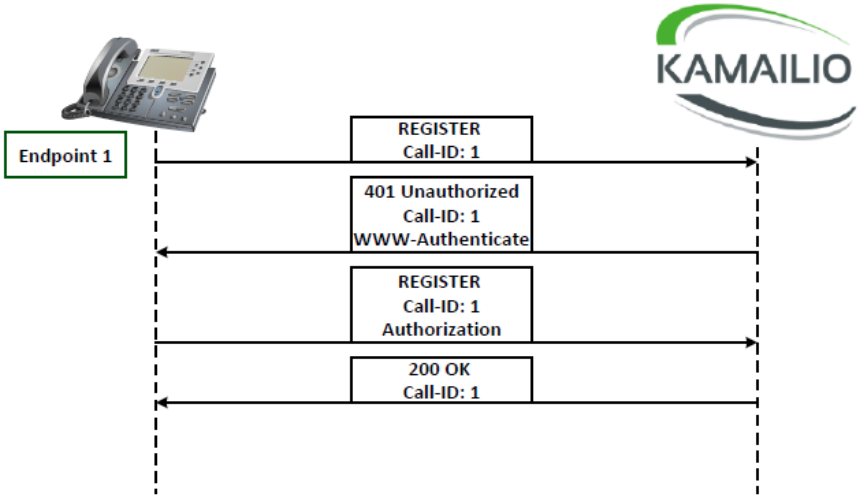


Figura 9.7: Flujo de Comunicación en el Registro de un Endpoint

9.3.5.3 Flujo de Tráfico Multimedia en Llamadas Establecidas

Una vez establecida una llamada (iniciada correctamente las sesiones INVITE como definido en la Figura 9.6), se entra en el proceso del envío de tráfico multimedia entre los endpoints. Aunque la misma podría lograrse, en ciertos casos, sin la intervención de Asterisk o Kamailio, gracias a la capacidad de transcoding de Asterisk y la posibilidad de grabación de llamadas, es común que el tráfico multimedia (y establecido por defecto) pase por el servidor de Asterisk.

Ahora, los usuarios registrados contra la arquitectura no tienen que obligatoriamente encontrarse dentro de la misma arquitectura de red donde la implementación fue formada, esto lo que conlleva al fenómeno de NAT Traversal anteriormente mencionado. La resolución del mismo es especificada más adelante en la configuración de Kamailio con la modificación de cabeceras a nivel de SIP y SDP y la comunicación mediante el socket de control con el proxy RTP.

El proxy RTP se encarga de la retransmisión del tráfico, basándose en las modificaciones de cabecera y la comunicación de Kamailio con RTPProxy, para permitir la comunicación entre Asterisk y los distintos endpoints en caso de que alguno de dichos endpoints se encuentre en otra arquitectura de red. Esta solución permite el escalamiento de distintos servidores de Asterisk en la arquitectura sin grandes modificaciones en elementos extras dentro de la arquitectura y la necesidad de múltiples direcciones públicas por cada nodo de Asterisk en caso de dejar la resolución de NAT Traversal a los servidores de Asterisk. La Figura 9.8 y la Tabla 9.3 explican el flujo de tráfico RTP en caso de escenarios que tengan que resolver problemas de NAT Traversal y aquellos que no lo requieran.

Comunicación	Endpoint 1	Endpoint 2
No NAT	No NAT	No NAT
NAT Traversal E1	NAT Traversal	No NAT
NAT Traversal E2	No NAT	NAT Traversal
NAT Traversal	NAT Traversal	NAT Traversal

Tabla 9.3: Relación entre los Distintos Escenarios de Flujos RTP

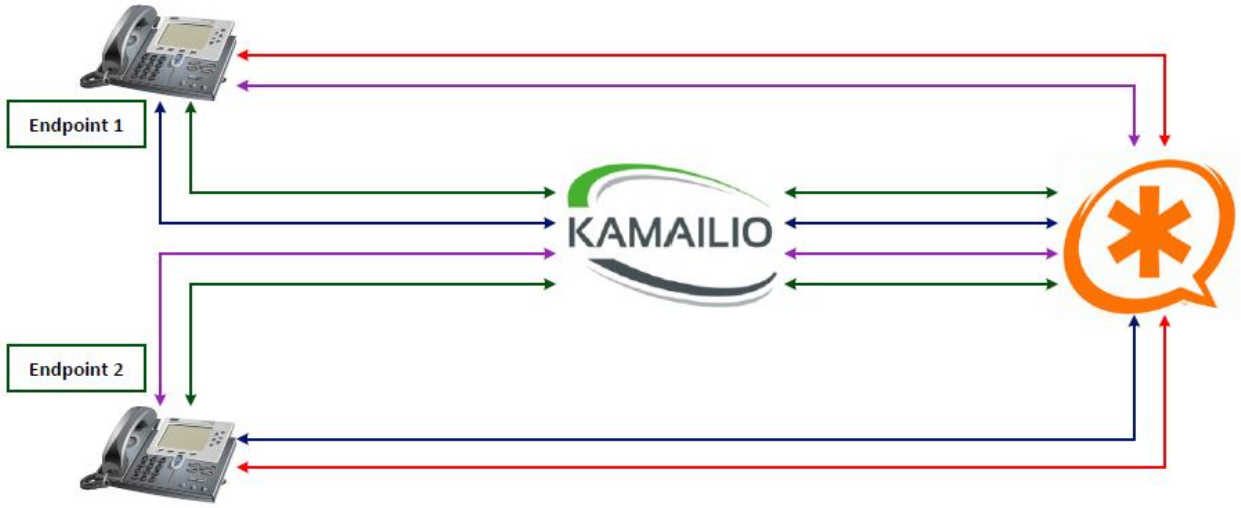


Figura 9.8: Flujo de Tráfico Multimedia en Llamadas Establecidas

9.4 Descripción de las Actividades de Implementación

En esta sección, se describe el proceso de las actividades realizadas para la configuración de la arquitectura de telefonía implementada. Algunas de las actividades realizadas serán explicadas con mayor detalle en este punto como justificación a algunas de las decisiones tomadas a la implementación de la arquitectura. Algunos de los puntos que serán tocados en esta sección son los siguientes:

- Consideraciones generales en instalaciones iniciales.
- Mecanismos de Fencing con el Hipervisor Xen.
- Configuración de Almacenamiento Compartido mediante Linux-IO iSCSI.
- Configuración de Componentes PBX.
- Configuración de Componentes Proxy/Registrar.
- Configuración de Escenarios de Prueba con SIPp.

9.4.1 Consideraciones Generales en Instalaciones Iniciales

Como proceso inicial, es importante que las máquinas que forman parte de la arquitectura tengan un cliente **NTP** instalado y sincronizado debido a que, para la formación de los esquemas de clustering con PaceMaker y Corosync, es indispensable este rubro.

Paquetes básicos

- yum install wget ntp net-tools vim bind-utils htop tcpdump ngrep
- ntpdate <servidor como ntp proxy o ntp server>

9.4.2 Mecanismos de Fencing con el Hipervisor Xen

Antes de la formación completa de los esquemas de clustering, es importante tener configurado los mecanismos de fencing que serán aplicados a dichos esquemas debido a las razones explicadas anteriormente.

Debido a la ausencia de dispositivos destinados a la aplicación de mecanismos de fencing, pero contando con que la implementación es montada en un entorno virtualizado utilizando un hipervisor como Xen, existen módulos que permiten que estos mecanismos de fencing sean monitoreados y realizados por el mismo hipervisor. Los módulos utilizados son **fence_virt v0.2.3** y **fence_xvm** que aprovechan un toolkit utilizado para interactuar en ambientes de virtualización para su gestión, principalmente con Xen y KVM, llamado **Libvirt v1.3.0**.

9.4.2.1 Configuración de Demonio fence_virt

Para lograr estos mecanismos, es necesario ingresar en el Dominio 0 del servidor donde fue instalado el hipervisor Xen Project. Ahí es configurado el demonio de fence_virt. El dominio 0 contiene el demonio de fencing de fence_virt, cuya configuración se muestra en la Figura 9.9.

```
10: fence_virtd {
11:         listener = "multicast";
12:         backend = "libvirt";
13:         module_path = "/usr/lib64/fence-virt";
14: }
15:
16: listeners {
17:         multicast {
18:                 key_file = "/etc/cluster/fence_xvm.key";
19:                 address = "225.0.0.12";
20:                 family = "ipv4";
21:                 port = "1229";
22:                 interface = "xenbr3";
23:         }
24: }
25:
26: backends {
27:         libvirt {
28:                 uri = "xen:///";
29:         }
30: }
```

Figura 9.9: Configuración del Demonio de Fencing

La Figura 9.9 especifica cómo queda el archivo de configuración destinado al demonio de fence_virt, algunos de sus parámetros significan:

- **fence_virtd:**
  - listener=”multicast”: la comunicación entre el dominio y los clientes que se encargan del monitoreo para mandar la señal de fencing será mediante mensajes multicast
  - backend=”libvirt”: define a Libvirt para la gestión de estas máquinas virtuales una vez interpretada las señales de fencing
- **listener:** Como especificado anteriormente, se utilizará multicast
  - key_file = Ruta de la llave que se utiliza para autenticar quienes desean la aplicación del fencing
  - address y port = Dirección IP y puerto de multicast por el cual se comunican
  - Interfaz = Interfaz por la cual recibe y envía los mensajes relacionados al fencing
- **Backends:** Es el componente que se encargará de gestionar las máquinas virtuales al recibir la señal de fencing
  - uri = Debido a que Libvirt puede ser utilizado con otros tipos de hipervisores (principalmente con KVM pero también con Xen), este se comunica con ellos mediante un URI. Aunque no hay ningún URI relevante para distinguir distintas instancias de un mismo tipo de hipervisor, es relevante distinguir el tipo de hipervisor, para KVM es “qemu:///” mientras que para Xen es “xen:///”

La llave especificada en la configuración de fencing debe ser creada y transferida a los distintos nodos que deseen formar parte de los mecanismos de fencing para la autenticación. Así, permitiendo mandar las señales de fencing al demonio de fence_virt para aplicar el mecanismo a otros guests (máquinas virtuales) que formen parte (éstos deben compartir la misma llave).
Una vez completado la configuración del demonio de fence_virt, es requerido instalar y configurar en las distintas máquinas virtuales los clientes de fencing que se encargan de enviar las señales de fencing.

9.4.2.2 Configuración de Cliente fence_virt

Junto con la configuración del demonio de fence_virt, es requerida la configuración de los clientes de fence_virt en los guests para encargarse de mandar las señales de fencing. Para ello, es necesario contener la llave, anteriormente mencionada, compartida entre los clientes y el demonio de fencing configurado en el Dominio 0. Con esto, se pueden mandar las señales al demonio para que Libvirt se encargue de aplicar la acción deseada al guest que se le desea aplicar dicha acción. Un ejemplo de su funcionamiento puede verse en el siguiente comando ejecutado desde el componente PBX 1 aplicando la acción al componente PBX 2 con el cliente de fence_virt configurado:

- fence_xvm –a 225.0.0.12 –k /etc/cluster/fence_xvm.key –H <nombre del guest> -ddd –o status

```
[root@pbx1 ~]# fence_xvm -a 225.0.0.12 -k /etc/cluster/fence_xvm.key -H tegamjg_pbx2  -ddd -o status
Debugging threshold is now 3
-- args @ 0x7ffe912ce750 --
  args->domain = tegamjg_pbx2
  args->op = 4
  args->mode = 0
  args->debug = 3
  args->timeout = 30
  args->delay = 0
  args->retr_time = 20
  args->flags = 0
  args->net.addr = 225.0.0.12
  args->net.ipaddr = (null)
  args->net.key_file = /etc/cluster/fence_xvm.key
  args->net.port = 1229
  args->net.hash = 2
  args->net.auth = 2
  args->net.family = 2
  args->net.ifindex = 0
  args->serial.device = (null)
  args->serial.speed = 115200,8N1
  args->serial.address = 10.0.2.179
-- end args --
Reading in key file /etc/cluster/fence_xvm.key into 0x7ffe912cd700 (4096 max size)
Actual key length = 4096 bytes
Sending to 225.0.0.12 via 127.0.0.1
Sending to 225.0.0.12 via 192.168.7.200
Sending to 225.0.0.12 via 10.0.1.200
Sending to 225.0.0.12 via 10.0.2.200
Sending to 225.0.0.12 via 10.0.3.200
Waiting for connection from XVM host daemon.
Issuing TCP challenge
Responding to TCP challenge
TCP Exchange + Authentication done...
Waiting for return value from XVM host
Status: ON
```

Figura 9.10: Resultado de la Aplicación de una Señal de Fencing

Con esto se culmina la configuración de los mecanismos de fencing. Sin embargo, para proveer mecanismos de contingencia en caso de errores dentro del esquema de clustering, es requerida la configuración del recurso encargado de aplicar fencing en los distintos esquemas de clustering como explicado más adelante.

9.4.3 Configuración de Almacenamiento Compartido mediante Linux-IO iSCSI

En el componente BD/SAN, se instala un servidor SAN mediante un programa que provee los servicios para la configuración de un iSCSI Target. La razón de esto, es la inclusión de un sistema de archivo compartido para ambas instancias de Asterisk. Esto permite tener los archivos de configuración compartidos entre ambas instancias, pero también mantener los archivos multimedia formados por múltiples servicios que puede proveer Asterisk (voicemail, call recording, etc) de manera compartida. Para ello, es

necesario configurar tanto un iSCSI Target como los distintos iSCSI Initiators que desean iniciar sesión con dicho Target.

9.4.3.1 Configuración de iSCSI Target

La configuración del iSCSI Target consiste en la configuración de los siguientes componentes que permiten definir al Target, las unidades lógicas y el método de acceso al mismo

- **iSCSI Qualified Name (IQN)**: un IQN consiste de ciertos campos como lo define el RFC 3720. Aunque no se considera de mucha importancia los detalles de los mismos en la implementación, es utilizado para definir unívocamente un iSCSI Target.
- **Target Portal Group (TPG):** TPG es un grupo que permite la creación de múltiples portales. Éste es configurado por defecto una vez definido la creación del Target.
- **Logical Unit Number (LUN):** Es la unidad lógica que agrupa el arreglo de discos al cual se desea acceder. Aquí se agrega la partición LVM creada previamente.
- **Portals:** Los listeners o portales son los controladores de escucha del Target. Éstos definen como acceder al Target. En este caso, se define una dirección y puerto por defecto que utiliza iSCSI.
- **Access Control List (ACL)**: Estas son reglas de acceso que se encargan de definir permisologías y los métodos de autenticación por las que los iSCSI Initiators (explicado más adelante) deben pasar para autenticarse y abrir una sesión con el Target. Se crea un ACL por cada uno de los componentes PBX que desean crear una sesión iSCSI con el iSCSI Target.

  El método de autenticación utilizado es CHAP, el cual es un método para ese efecto empleado por iSCSI para certificar a los distintos Initiators que deseen acceder al Target.

La configuración del iSCSI Target queda, como mostrado en la Figura 9.11.

```
/> ls
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block .................................................. [Storage Objects: 1]
  | | o- disk1 ............... [/dev/vg_san1/lv_pbx (19.9GiB) write-thru activated]
  | o- fileio ................................................. [Storage Objects: 0]
  | o- pscsi .................................................. [Storage Objects: 0]
  | o- ramdisk ................................................ [Storage Objects: 0]
  o- iscsi ............................................................ [Targets: 1]
  | o- iqn.2016-06.10.0.2.202:storage.target01 ........................ [TPGs: 1]
  |   o- tpg1 ............................................... [no-gen-acls, no-auth]
  |     o- acls ...................................................... [ACLs: 2]
  |     | o- iqn.2016-06.10.0.2.202:pbx1.example.com ............ [Mapped LUNs: 1]
  |     | | o- mapped_lun0 ................................ [lun0 block/disk1 (rw)]
  |     | o- iqn.2016-06.10.0.2.202:pbx2.example.com ............ [Mapped LUNs: 1]
  |     |   o- mapped_lun0 ................................ [lun0 block/disk1 (rw)]
  |     o- luns ...................................................... [LUNs: 1]
  |     | o- lun0 ............................. [block/disk1 (/dev/vg_san1/lv_pbx)]
  |     o- portals ................................................ [Portals: 1]
  |       o- 10.0.2.202:3260 ..................................... [OK]
  o- loopback ......................................................... [Targets: 0]
/iscsi/iqn.20...1.example.com> ls
o- iqn.2016-06.10.0.2.202:pbx1.example.com ....................... [Mapped LUNs: 1]
  o- mapped_lun0 ........................................... [lun0 block/disk1 (rw)]
/iscsi/iqn.20...1.example.com> info
chap_password: sys64738_
chap_userid: asterisk
wwns:
iqn.2016-06.10.0.2.202:pbx1.example.com
/iscsi/iqn.20...1.example.com> cd ../iqn.2016-06.10.0.2.202:pbx2.example.com/
/iscsi/iqn.20...2.example.com> ls
o- iqn.2016-06.10.0.2.202:pbx2.example.com ....................... [Mapped LUNs: 1]
  o- mapped_lun0 ........................................... [lun0 block/disk1 (rw)]
/iscsi/iqn.20...2.example.com> info
chap_password: sys64738_
chap_userid: asterisk
wwns:
iqn.2016-06.10.0.2.202:pbx2.example.com
```

Figura 9.11: Configuración de iSCSI Target

9.4.3.2 Configuración de iSCSI Initiator

Como fue referido anteriormente, se considera que los componentes con Asterisk sean los que utilicen este almacenamiento compartido. Por ello, se definen éstos como iSCSI Initiators. Un iSCSI Initiator es una instancia que se encarga de iniciar una sesión iSCSI para que el iSCSI Target provea uno o más LUNs definidos en ese Target. Así, el Initiator podrá gestionar, mediante comandos de escritura o lectura, la transferencia de datos requerida.

Cada iSCSI Initiator debe pasar por el mecanismo de autenticación CHAP definidos en el iSCSI Target. Por ello se define el nombre del ACL creado en el iSCSI Target y el usuario y contraseña que es requerido para ese ACL para lograr la autenticación.

- **Initiator Name:** Nombre del ACL para definir a cual regla se procede para el proceso de autenticación.

```
1: InitiatorName=iqn.2016-06.10.0.2.202:pbx1.example.com
```

Figura 9.12: Definición de la Identificación del Initiator

- **Configuración de CHAP:** En la Figura 9.13, se define el usuario (línea 8) y contraseña (línea 9) para comenzar el proceso de autenticación

```
2: # To enable CHAP authentication set node.session.auth.authmethod
3: # to CHAP. The default is None.
4: node.session.auth.authmethod = CHAP
5:
6: # To set a CHAP username and password for initiator
7: # authentication by the target(s), uncomment the following lines:
8: node.session.auth.username = asterisk
9: node.session.auth.password = sys64738_
```

Figura 9.13: Definición de Parámetros para la Autenticación mediante CHAP

En caso de ser exitosa la autenticación, se crea la sesión iSCSI y ya se puede ver el acceso al almacenamiento definido en el iSCSI Target como mostrado en la Figura 9.14.

```
Disk /dev/sda: 21.4 GB, 21369978880 bytes, 41738240 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 131072 bytes
```

Figura 9.14: Partición tomada desde el iSCSI Target

Con lo definido en esta sección, se tiene configurado el acceso al almacenamiento compartido. Pero, como comentado anteriormente, es requerido la configuración del esquema de clustering al momento del formateo de dicha partición compartida. En el próximo punto se comenta acerca de la configuración de los componentes PBX.

9.4.4 Configuración de Componentes PBX

En este punto, se tocan todas las configuraciones por las cuales pasan ambas instancias para lograr lo especificado en los capítulos anteriores, algunos de los aspectos a tocar en este punto son:

- Asterisk Realtime Architecture.
- Personalización del Agente de Recurso OCF de Asterisk.
- Creación de Esquema de Clustering con Componentes PBX.
- Migración de Contenido de Asterisk.
- Configuración General de Asterisk.

9.4.4.1 Asterisk Realtime Architecture

Asterisk por defecto no almacena información de configuración en base de datos, sino que todo lo maneja en archivos de configuración. Por ello, para mantener el almacenamiento de manera compartida se opta por configurar dicha arquitectura.

Primero se debe tener claro el esquema de la base de datos de Asterisk; las tablas relevantes en las siguientes tablas mostradas en la Figura 9.15.

```
mysql> show tables;
+----------------------+
| Tables_in_asterisk_db |
+----------------------+
| sipregs              |
| sipusers             |
| voicemail            |
+----------------------+
3 rows in set (0.00 sec)
```

Figura 9.15: Esquema de Base de Datos de Asterisk

Las tablas que terminaron siendo utilizadas son **sipusers** y **voicemail**, debido a que el registro de usuarios es gestionado por Kamailio. La ventaja de mantenerlos en base de datos, es la posibilidad de gestionar a los usuarios y buzones de voz en tiempo real y, además, es muy útil en caso de tener múltiples instancias de Asterisk que requieran compartir esta misma información de configuración. Otra posibilidad que ofrece este tipo de arquitecturas, es la agregación del Dialplan en base de datos. Sin embargo, no terminó siendo considerado, debido a que su gestión y mantenimiento es muy ineficiente.

Ahora, para ofrecer este tipo de arquitecturas, se debe configurar en Asterisk el acceso sobre la base de datos mediante la configuración de **unixODBC v2.3.1**.

- **ODBC Drivers**: Se especifica la ubicación de los drivers ODBC para la comunicación con un sistema manejador MySQL. Su configuración se ve en la Figura 9.16.

```
11: [MySQL]
12: Description=ODBC for MySQL
13: Driver=/usr/lib/libmyodbc5a.so
14: Setup=/usr/lib/libodbcmyS.so
15: Driver64=/usr/lib64/libmyodbc5a.so
16: Setup64=/usr/lib64/libodbcmyS.so
17: FileUsage=1
```

Figura 9.16: Configuración de Ubicación de Drivers para MySQL

- **ODBC Instance**: Se especifica el Data Source Name (DSN) que define la configuración para comunicarse con Asterisk. Su configuración se ve en la

```
10: [Asterisk-MySQL]
11: Description = Asterisk Realtime
12: Driver = MySQL
13: Trace = Off
14: TraceFile = stderr
15: SERVER = db1vs2
16: USER = userdb
17: PASSWORD = $Sys64738_
18: PORT = 3306
19: DATABASE = asterisk_db
```

Figura 9.17.

Figura 9.17: Configuración del Data Source Name

- **Recurso de ODBC de Asterisk:** Informa a Asterisk los parámetros y la configuración del DSN para comunicarse con la base de datos. Su configuración se ve en la Figura 9.18.

```
1: [asterisk_db]
2: enabled => yes
3: dsn => Asterisk-MySQL
4: username => userdb
5: password => $Sys64738_
6: pre-connect => yes
```

Figura 9.18: Configuración de ODBC en Asterisk

- **Especificación de servicios de Asterisk a utilizar por base de datos**: Se especifica que servicios serán gestionadas por esta arquitectura. Su configuración se ve en la Figura 9.19.

```
1: [settings]
2: sipusers => odbc,asterisk_db,sipusers
3: sippeers => odbc,asterisk_db,sipusers
4: sipregs => odbc,asterisk_db,sipregs
5: voicemail => odbc,asterisk_db,voicemail
```

Figura 9.19: Servicios de Asterisk gestionados por Base de Datos

Con lo especificado en este punto, se tiene configurado Asterisk Realtime Architecture. Sin embargo, no toda la información requerida es almacenada de manera compartida solo con esta arquitectura. Es por ello, que surge la necesidad de la configuración del almacenamiento compartido como especificado en el punto 9.4.3. Pero, para poner en producción dicho almacenamiento compartido se debe crear el esquema de clustering.

9.4.4.2 Personalización del Agente de Recurso OCF de Asterisk

Para el monitoreo del servicio de Asterisk, se cuenta con un recurso OCF que se encarga de la gestión del servicio de Asterisk. Para esta arquitectura, se propone la modificación de este recurso, principalmente en la función de monitoreo, debido a que originalmente el monitoreo consta (además de monitorear el estado de los procesos, funcionalidad del CLI, etc) del envío de mensajes OPTION utilizando **SIPSAK** hacia Asterisk, esperando una respuesta por parte de Asterisk.

Debido a que Asterisk tiene la función principal en esta implementación de prestar servicios de VoIP multimedia, es importante también monitorear el estado de módulos que forman parte esencial del core de Asterisk para su funcionalidad con Real Time Protocol (aplicaciones como playback, interpretación de tonos DTMF etc). Por ello, se creó un escenario con SIPp, más la configuración de Asterisk, para la realización de una llamada hacia un IVR que espera el ingreso de tonos DTMF, simular dichos tonos DTMF mediante SIPp que llevan a un Playback en el Plan de Discado en Asterisk. Esto, permite asegurar que el módulo de SIP y RTP están cargados y funcionales, como además la interpretación de tonos DTMF y ciertas aplicaciones como las de playback y background.
La estructura de la creación de escenarios con SIPp es explicada con más detalle en el punto 9.4.6. Por ello, se hará una cierta mención poniendo el acento en la lógica del flujo de monitoreo, como mostrado en la Figura 9.20.

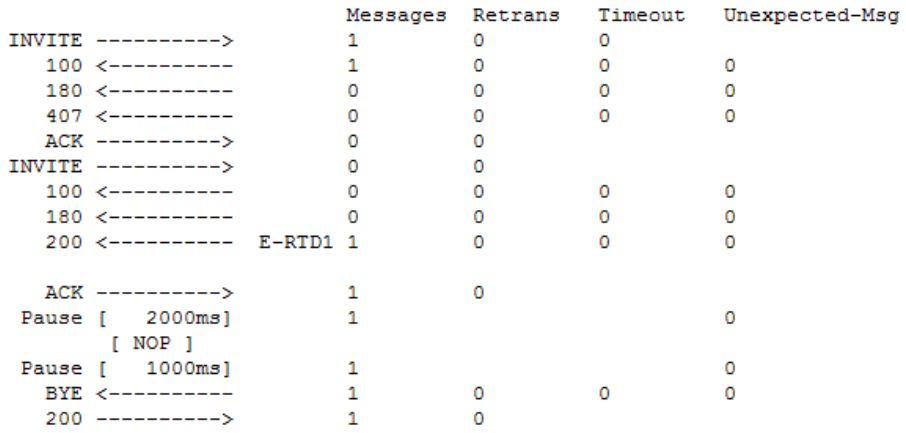


Figura 9.20: Flujo de Mensajes de Monitoreo del Recurso de Asterisk

Es importante resaltar que un mensaje BYE por parte de Asterisk no es siempre un mensaje de culminación positiva. Asterisk maneja ciertos códigos como una adaptación de los códigos que maneja Integrated Services Digital Network (ISDN) para la representación de código de salida de una llamada. Asterisk representa su culminación de una llamada exitosa con un mensaje BYE que contiene una cabecera extra (llamada X-Asterisk-HangupCauseCode) con valor de 16, esto indica una culminación de llamada exitosa (Normal Clearing). En caso de ser cualquier otro valor es posible considerarlo como un error en el proceso y será percibida por SIPp como un error. Lo expresado puede verse en anteriormente la línea 21 de la Figura 9.21 que refleja el escenario creado con SIPp como se explicará más adelante, en esta se ubica que la cabecera X-Asterisk-HangupCauseCode en el mensaje BYE exista y contenga el valor

```
10:  <pause milliseconds="2000"/>
11:  <!-- Play an out of band DTMF '2'                                    -->
12:  <nop>
13:    <action>
14:      <exec play_pcap_audio="pcap/dtmf_2833_2.pcap"/>
15:    </action>
16:  </nop>
17:  <pause milliseconds="1000"/>
18:
19:  <recv request="BYE">
20:        <action>
21:                <ereg regexp=" 16$" search_in="hdr" header="X-Asterisk-
   HangupCauseCode:" check_it="true" assign_to="3" />
22:        </action>
23:  </recv>
24:  <Reference variables="3"/>
```

16:

Figura 9.21: Códigos de Salida de Asterisk Expresados en el Escenario SIPp

Aplicación del monitoreo personalizado en el RA de Asterisk

La Figura 9.22 muestra la función creada que se encarga de llamar, mediante la ejecución de un bash script (línea 15), el escenario de SIPp que se encarga de monitorear el estado de Asterisk.

```
10: asterisk_sipp() {
11:         modulecheck=`asterisk -rx "module show like chan_sip.so" |grep
    "chan_sip.so" |wc -l`
12:         sipcheck=`asterisk -rx "sip show peers" |grep "No such command" |wc -l`
13:         if [ $modulecheck -eq 1 ] && [ $sipcheck -eq 0 ]; then
14:                 ocf_log info "UBICAR: SIP module is active and running"
15:                 sh $OCF_RESKEY_sipp_monitor &> /dev/null
16:                 rc=$?
17:                 case "$rc" in
18:                  1|2) return $OCF_ERR_GENERIC;;
19:                  3)   return $OCF_NOT_RUNNING;;
20:                 esac
21:         else
22:                 ocf_log err "UBICAR: SIP module is still not running"
23:                 return $OCF_NOT_RUNNING;
24:         fi
25:         return $OCF_SUCCESS
26: }
```

Figura 9.22: Función Encargada de Monitorear a Asterisk mediante SIPp

Para los proceso de monitoreo del recurso, la función reflejada en la Figura 9.22 se encuentra dentro de un bucle reintentando el proceso mientras resulte en error hasta que el rango de tiempo cumpla (valor de interval en el recurso de Asterisk, especificado en la Sección 9.4.4.3). En el proceso de inicio del recurso, esta función es llamada una única vez y deberá responder de manera correcta para iniciar.

Con la personalización del recurso OCF completada, se puede proceder a la creación del esquema de clustering, algunos parámetros extras son agregados al recurso de OCF para poder especificar aspectos relacionados al monitoreo personalizado en este recurso.

9.4.4.3 Creación de Esquema de Clustering con Componentes PBX

Como antes fue indicado, se desea crear un cluster Activo/Activo con Asterisk. Para ello se utiliza Pacemaker y Corosync, gestionados por PCSD. PCSD es una interfaz de línea de comandos que permite la gestión y creación de esquemas de clustering.

PCSD crea un usuario por defecto en el sistema (**hacluster**) para la creación y gestión del cluster. Es necesario que ambos nodos que están formando parte del cluster tengan la misma contraseña para este usuario para lograr autenticarse entre ellos y permitir la creación del cluster. La creación del mismo forma la configuración básica de Corosync, mostrada en la Figura 9.23, la modificación de dicha configuración solo trata de la inclusión de un archivo de log (línea 36).

```
totem {
    version: 2
    secauth: off
    cluster_name: asteriskcluster
    transport: udpu
}
nodelist {
    node {
        ring0_addr: pbx1vs3
        nodeid: 1
    }
    node {
        ring0_addr: pbx2vs3
        nodeid: 2
    }
}
quorum {
    provider: corosync_votequorum
    two_node: 1
}
logging {
    to_logfile: yes
    logfile: /var/log/cluster/corosync.log
    to_syslog: no
}
```

Figura 9.23: Configuración de Corosync

Antes de la definición de Pacemaker, es necesario crear un LVM en cada nodo perteneciente al cluster al bloque que fue montado por el Initiator al autenticarse con el iSCSI Target. La definición del mismo queda como en la Figura 9.24.

```
Disk /dev/sda: 21.4 GB, 21369978880 bytes, 41738240 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 131072 bytes


Disk /dev/mapper/vg_san1-lv_pbx: 21.3 GB, 21260926976 bytes, 41525248 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 131072 bytes
```

Figura 9.24: Resultado de la Creación del Volumen Lógico

Es importante destacar que el LVM creado debe estar consciente de que se encuentra en un esquema de clustering que, en combinación con Clustered Logical Volume (cLVM) gestiona los bloqueos requeridos para evitar la corrupción de información de los volúmenes que gestiona. Este volumen lógico luego es formateado con el sistema de archivo GFS2 que, como comentado anteriormente, es requerido en ambientes de clustering Activo/Activo donde se requiera el acceso y el montaje de dicho sistema de archivo en ambos nodos. La definición del sistema de archivo GFS2 puede verse en la Figura 9.25.

```
[root@pbx1 ~]# df -iTH
Filesystem                Type      Inodes IUsed IFree IUse% Mounted on
/dev/xvdb1                ext4        1.4M   76k  1.3M    6% /
devtmpfs                  devtmpfs    231k   396  230k    1% /dev
tmpfs                     tmpfs       233k   141  233k    1% /dev/shm
tmpfs                     tmpfs       233k   474  233k    1% /run
tmpfs                     tmpfs       233k    13  233k    1% /sys/fs/cgroup
/dev/xvda1                ext4         66k   345   66k    1% /boot
/dev/xvdc1                ext4        132k    16  132k    1% /home
/dev/mapper/vg_san1-lv_pbx gfs2       5.1M   12k  5.1M    1% /mnt/asterisk
tmpfs                     tmpfs       233k     1  233k    1% /run/user/0
```

Figura 9.25: Definición del Sistema de Archivo GFS2

Para el formateo del sistema de archivo GFS2 se corre el siguiente comando:

- mkfs.gfs2 -p lock_dlm -j 2 -t asteriskcluster:asteriskfs /dev/vg_san1/lv_pbx (creación del sistema de archivo GFS2 en dicha partición)
  - lock_dlm: define el módulo de DLM que se encarga de la gestión de permisologias, éste debe estar cargado previamente
  - –j 2: Es para indicar que se utilizarán 2 journals, uno para cada nodo perteneciente al cluster. Como en sistemas de archivos ext3, GFS2 escribe la metadata en el journal antes de ponerlo en su lugar. Esto permite la recuperación de la metadata en caso, por ejemplo, de cortes de luz
  - asteriskcluster:asteriskfs: El primer parámetro es el nombre del cluster creado con Pacemaker/Corosync, y asteriskfs es el nombre del recurso Filesystem que se encarga de gestionar el sistema de archivo (este es creado como un recurso en Pacemaker como se ve más adelante).

De aquí en adelante, se explica cómo queda la configuración del esquema de clustering de componentes PBX.

Nodos Pertenecientes al Esquema de Clustering

```
Cluster Name: asteriskcluster
Corosync Nodes:
 pbx1vs3 pbx2vs3
Pacemaker Nodes:
 pbx1vs3 pbx2vs3
```

Figura 9.26: Nodos Pertenecientes al Esquema de Clustering de Componentes PBX

La Figura 9.26 resalta los siguientes aspectos:

- **Cluster Name:** Nombre del cluster.
- **Corosync Nodes:** Aquellos nodos que forman parte en la infraestructura de Corosync. En este caso se pueden ver solo dos nodos (Activo/Activo).
- **Pacemaker Nodes:** Aquellos nodos donde se desean que se gestionen recursos (servicios). En este caso se pueden ver solo dos nodos (Activo/Activo).

Configuración de los Recursos Monitoreados

La Figura 9.27 muestra la configuración de los distintos recursos del esquema de clustering.

```
Resources:
 Clone: dlm-clone
  Meta Attrs: clone-max=2 clone-node-max=1 interleave=true
  Resource: dlm (class=ocf provider=pacemaker type=controld)
   Attributes: allow_stonith_disabled=false
   Operations: start interval=0s timeout=90 (dlm-start-interval-0s)
               monitor interval=10s on-fail=fence (dlm-monitor-interval-10s)
               stop interval=0s on-fail=fence (dlm-stop-interval-0s)
 Clone: clvmd-clone
  Meta Attrs: clone-max=2 clone-node-max=1 interleave=true
  Resource: clvmd (class=ocf provider=heartbeat type=clvm)
   Operations: start interval=0s timeout=90 (clvmd-start-interval-0s)
               monitor interval=10s on-fail=fence (clvmd-monitor-interval-10s)
               stop interval=0s on-fail=fence (clvmd-stop-interval-0s)
 Clone: asteriskfs-clone
  Meta Attrs: interleave=true clone-max=2 clone-node-max=1
  Resource: asteriskfs (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=/dev/vg_san1/lv_pbx directory=/mnt/asterisk fstype=gfs2
   Operations: start interval=0s timeout=60 (asteriskfs-start-interval-0s)
               monitor interval=10s on-fail=fence (asteriskfs-monitor-interval-10s)
               stop interval=0s on-fail=fence (asteriskfs-stop-interval-0s)
 Clone: asterisk-clone
  Meta Attrs: interleave=true ordered=true clone-max=2 clone-node-max=1
  Resource: asterisk (class=ocf provider=heartbeat type=asterisk)
   Attributes: user=root group=root config=/mnt/asterisk/etc/asterisk.conf maxfiles=65535
 sipp_monitor=/root/scripts/haasterisk.sh sipp_binary=/usr/local/src/sipp-3.4.1/bin/sipp
   Operations: monitor interval=10s on-fail=restart (asterisk-monitor-interval-10s)
               stop interval=0s on-fail=fence (asterisk-stop-interval-0s)
               start interval=0s timeout=40s (asterisk-start-interval-0s)
 Clone: fence_pbx1_xvm-clone
  Meta Attrs: interleave=true clone-max=2 clone-node-max=1
  Resource: fence_pbx1_xvm (class=stonith type=fence_xvm)
   Attributes: port=tegamjg_pbx1 pcmk_host_list=pbx1vs3
   Meta Attrs: resource-stickiness=200
   Operations: monitor interval=60s (fence_pbx1_xvm-monitor-interval-60s)
 Clone: fence_pbx2_xvm-clone
  Meta Attrs: interleave=true clone-max=2 clone-node-max=1
  Resource: fence_pbx2_xvm (class=stonith type=fence_xvm)
   Attributes: port=tegamjg_pbx2 pcmk_host_list=pbx2vs3
   Meta Attrs: resource-stickiness=200
   Operations: monitor interval=60s (fence_pbx2_xvm-monitor-interval-60s)
```

Figura 9.27: Configuración de los Recursos Gestionados por el cluster de Componentes PBX

En el esquema de clustering de componentes PBX se definieron 6 recursos clonados, 2 de ellos son dispositivos STONITH (fencing). Los detalles de cada recurso se describen a continuación:

- **dlm:** Recurso creado para iniciar, monitorear y parar el componente DLM. La utilidad de DLM fue descrita anteriormente, el mismo debe estar cargado antes de montar el sistema de archivo.
  - allow_stonith_disabled=false: Debe estar STONITH activo en el cluster (stonith-enabled=true) para iniciar el recurso.
  - monitor: monitoreo cada 10 segundos (interval=10s), en caso de fallar aplicar fencing en este nodo (on-fail=fence).
  - stop: en caso de fallar aplicar fencing en este nodo (on-fail=fence).
- **clvmd:** Recurso creado para iniciar, monitorear y parar el componente cLVM. También es requerido para la gestión de permisologias y bloqueo del acceso a datos concurrentemente.
  - monitor: monitoreo cada 10 segundos (interval=10s), en caso de fallar aplicar fencing en este nodo (on-fail=fence).
  - stop: en caso de fallar aplicar fencing en este nodo (on-fail=fence).

- **asteriskfs:** Recurso creado para iniciar (montar), monitorear y parar (desmontar) el sistema de archivo especificado.
    - device=/dev/vg_san1/lv_pbx: Dispositivo donde está formateado la partición que se desea montar.
    - directory=/mnt/asterisk: Ruta donde se desea montar dicha partición.
    - fstype=gfs2: Tipo de sistema de archivo que se desea montar, al colocar gfs2 obliga a verificar la existencia del componente DLM para ser montado.
    - monitor: monitoreo cada 10 segundos (interval=10s), stop: en caso de fallar aplicar fencing en este nodo (on-fail=restart).
- **asterisk:** Recurso creado para iniciar, monitorear y parar el servicio de Asterisk.
    - user=root group=root: Usuario/grupo que se encarga de iniciar el servicio.
    - config=/mnt/asterisk/etc/asterisk.conf: Ruta donde se encuentra el archivo de configuración general de Asterisk.
    - maxfiles=65535: Ampliación en la cantidad de Open File Descriptors que pueden ser gestionados por el proceso de Asterisk. Por ello, es necesario aumentar tanto el FD del proceso de Asterisk que, por defecto, es de un máximo de 2048, como el del OS para realizar las pruebas de estrés.
    - sipp_monitor=/root/scripts/haasterisk.sh: Script utilizado para monitorear el estado del servicio mediante una simulación de una llamada utilizando SIPp. Esto representa una personalización del agente de recurso de Asterisk.
    - sipp_binary=/usr/local/src/sipp-3.4.1/bin/sipp: Ubicación del binario de SIPp. Esto representa una personalización del agente de recurso de Asterisk.
    - monitor: monitoreo cada 10 segundos (interval=10s), en caso de fallar se reinicia el recurso (on-fail=restart).
    - stop: en caso de fallar aplicar fencing en este nodo (on-fail=fence).
- **fence_pbx1_xvm:** Recurso creado para monitorear la necesidad de mandar una señal al hipervisor para aplicar el fencing a Asterisk 1.
    - port=tegamjg_pbx1: Nombre de la máquina virtual a la cual se le aplica fencing.
    - pcmk_host_list=pbx1vs3: Lista de máquinas controladas por este dispositivo de fencing.
- **fence_pbx2_xvm:** Recurso creado para monitorear la necesidad de mandar una señal al hipervisor para aplicar el fencing a Asterisk 2.
    - port=tegamjg_pbx2: Nombre de la máquina virtual a la cual aplicarle fencing.
    - pcmk_host_list=pbx2vs3: Lista de máquinas controladas por este dispositivo de fencing.

Como se puede notar, todos los recursos clonados tienen unos meta atributos especificados que representan el comportamiento de los recursos clonados. Estos meta atributos son:

- **interleave=true**: Cuando hay una restricción de orden (ordering constraint) en el inicio de distintos recursos donde el que debe iniciar primero es un recurso clonado. Esta restricción obliga a que todas las instancias del recurso inicien en todos los nodos antes de poder proseguir con el próximo recurso, al colocar interleave=true solo está obligado a iniciar en ese nodo para poder iniciar el próximo recurso. El tener interleave=false puede conllevar a más delay para el inicio los servicios.
- **clone-max=2:** La máxima cantidad de veces que una misma instancia de un recurso puede iniciar en el cluster, se coloca 2 debido a que se desea una instancia en cada nodo.

- **clone-node-max=1:** La máxima cantidad de instancias de un mismo recurso que puede iniciar en un mismo nodo. Debido a que solo se desea una instancia de un mismo recurso por nodo se especifica 1.

Restricciones de los Recursos

La Figura 9.28 muestra la configuración de las restricciones de los distintos recursos que forman parte del esquema de clustering de componentes PBX:

```
Location Constraints:
Ordering Constraints:
  Resource Sets:
    set fence_pbx1_xvm-clone fence_pbx2_xvm-clone sequential=false set dlm-clone clvmd-clone
asteriskfs-clone asterisk-clone sequential=true
Colocation Constraints:
  clvmd-clone with dlm-clone (score:INFINITY)
  asteriskfs-clone with clvmd-clone (score:INFINITY)
  asterisk-clone with asteriskfs-clone (score:INFINITY)
```

Figura 9.28: Configuración de Restricciones de los Recursos Gestionados

Ciertas restricciones son aplicadas en los recursos del cluster. Dos tipos de restricciones son utilizadas aquí:

- **Ordering Constraints:** Esta restricción es utilizada para aplicar el orden en que los recursos de un cluster deben iniciar. La Figura 9.29 describe la restricción de orden:

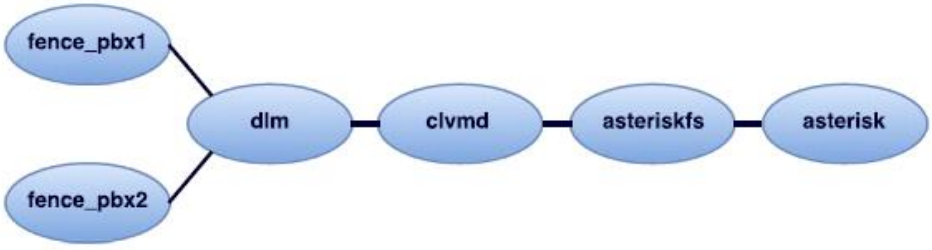


Figura 9.29: Restricción de Orden del Esquema de Clustering de Componentes PBX

Como lo muestra la Figura 9.28, la especificación de “Resource Sets” es una manera simplificada de configurar restricciones de ordenamiento, pero permite definir también restricciones más complejas. En este caso, los dispositivos de fencing pueden ser iniciados de manera no secuencial, pero una vez empezadas ambas, se prosigue con las restricciones. Esto es logrado colocando el parámetro “sequential=false” en un bloque set como lo muestra la Figura 9.28.

En este caso, estas restricciones son obligatorias (kind: Mandatory) para el inicio de los recursos en el orden especificado. Esto significa que, sin excepción, los recursos deben iniciar en el orden especificado. De la misma manera esta restricción obliga, en caso de requerir parar los recursos, a parar los recursos en el orden adverso, por ejemplo, si se desean parar todos los recursos de un nodo, primero debe parar Asterisk, luego el sistema de archivo y así sucesivamente.

- **Colocation Constraints:** Este tipo de restricción es utilizada para aplicar una limitación sobre los recursos para que un recurso no inicie en un nodo donde otro recurso no se encuentra. En esta implementación, su utilidad es iniciar los recursos en un mismo nodo, pueden ser en ambos al ser un cluster Activo/Activo. Aunque útil en esquemas Activo/Activo es mayormente utilizado en esquemas Activo/Pasivo.

Propiedades a nivel del Cluster

La Figura 9.30 muestra la configuración de las propiedades generales a nivel del esquema de clustering de componentes PBX.

```
Resources Defaults:
 migration-threshold: 2
 failure-timeout: 10m
 start-failure-is-fatal: false
 resource-stickiness: 200
Operations Defaults:
 No defaults set

Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: asteriskcluster
 dc-version: 1.1.13-10.el7_2.2-44eb2dd
 have-watchdog: false
 last-lrm-refresh: 1470623056
 no-quorum-policy: ignore
 start-failure-is-fatal: false
 stonith-action: reboot
 stonith-enabled: true
```

Figura 9.30: Propiedades a Nivel del Esquema de Clustering de Componentes PBX

Se pueden ver dos tipos de propiedades especificadas:

- **Resources Defaults:** Representa la configuración a nivel de recursos pero general. Este afecta a todos los recursos, salvo que se especifique ese mismo parámetro en el recurso específicamente mediante un meta atributo
  - migration-threshold: 2: Este parámetro especifica que un recurso que haya alcanzado una cantidad de fallas (en este caso principalmente de monitoreo) parará ese recurso y no podrá correr dicho recurso nuevamente salvo que se limpie este contador, se reinicie el cluster o se cumpla con el timer especificado en **failure-timeout.**
  - failure-timeout: 10m: Como dicho en el párrafo anterior, este parámetro es especificado para resetear los contadores de falla de los recursos dentro del cluster. Así, permitiendo el reinicio de los recursos en ese nodo en caso de ser aplicable.
  - start-failure-is-fatal: false: Por defecto, cuando hay un fallo en el inicio/parado de un recurso, el contador de fallas de ese recurso se coloca en INFINITY, lo que evita el reintento en su inicio/parado. Colocando este parámetro en falso, aplica a esta falla como un contador más de la misma manera como una falla en caso de monitoreo.
  - resource-stickiness: 200: Muy utilizado para evitar migraciones de recursos a otros nodos si en el nodo que se encuentra está totalmente “saludable” (sin fallas en los recursos que gestiona). Estas migraciones que se desean evitar suelen ser migraciones no deseadas.
- **Cluster Properties:** Propiedades generales del cluster
  - no-quorum-policy: ignore: Quorum es una funcionalidad muy útil ofrecida, en este caso, por Corosync, pero principalmente en clusters que constan de más de dos nodos, por ello su política es ignorada en este caso.

- stonith-action: reboot: Especifica el tipo de señal que envía los recursos de fencing configurados en caso de requerirlo. Debido a que el mecanismo de fencing mediante fence_virtd no ofrece la posibilidad de aplicar fencing a través de shutdown, se aplica la de reinicio forzado del servidor.
- stonith-enabled: true: Para habilitar a los dispositivos de STONITH. En caso de encontrarse activo y no detectar ningún dispositivo de STONITH o fencing disponible el cluster no iniciará.

Con esta configuración, se logra la creación del esquema de clustering con los aspectos deseados y descritos en la Sección 9.3.4.2.

9.4.4.4 Migración del Contenido de Asterisk

Es preferible y, en ciertos casos, necesario que algunos de los directorios donde se encuentran los archivos de Asterisk se coloquen en la ruta donde está la partición compartida. Esto para su acceso concurrente por ambas instancias. En la Figura 9.31 se puede ver cómo queda la configuración:

```
10: [directories]
11: astetcdir => /mnt/asterisk/etc
12: astmoddir => /usr/lib64/asterisk/modules
13: astvarlibdir => /var/lib/asterisk
14: astdbdir => /mnt/asterisk/lib
15: astkeydir => /mnt/asterisk/lib
16: astdatadir => /mnt/asterisk/lib
17: astagidir => /mnt/asterisk/lib/agi-bin
18: astspooldir => /mnt/asterisk/spool
19: astrundir => /var/run/asterisk
20: astlogdir => /var/log/asterisk
21: astsbindir => /usr/sbin
```

Figura 9.31: Configuración de Ubicación de Contenido de Asterisk

- **astetcdir => /mnt/asterisk/etc:** mantener los archivos de configuración en nodos que desean operar igual siempre es buena idea.
- **astdbdir => /mnt/asterisk/lib**
- **astkeydir => /mnt/asterisk/lib**
- **astdatadir => /mnt/asterisk/lib:** Este es el directorio de los datos de Asterisk como archivos de music on hold (MOH) y sonidos en general que se deseen grabar o los que vienen por defecto.
- **astagidir => /mnt/asterisk/lib/agi-bin**
- **astspooldir => /mnt/asterisk/spool:** En este directorio se guardan los archivos multimedia generados por mensajes de voz, además como grabación de llamadas, por ende es necesario mantenerlo compartido así ambos nodos puedan tener acceso en tiempo real a las modificaciones hechas por cada uno.

Ya se tienen la información de Asterisk requerida compartida entre ambas instancias, de aquí en adelante, en este punto, se procede a describir el proceso de configuración del servicio de Asterisk.

9.4.4.5 Configuración de Asterisk

Aquí se dividen las configuraciones entre aquellas que se realizan a nivel de base de datos y aquellas que son realizadas en los archivos de configuración de Asterisk.

Base de Datos

- **sipusers:** Tabla utilizada para almacenar de manera compartida a los usuarios SIP, es importante destacar que Asterisk no se encarga del registro de los mismos. Sin embargo, eso no significa que no sea conveniente que Asterisk tenga conocimiento de dichos usuarios, debido a que permite la definición de contextos (Kamailio no maneja el término de contextos) y definición de buzones de mensajes por usuario.

```
mysql> select name,defaultuser,host,type,context,dtmfmode,insecure,mailbox,hasvoicemail,subscribemwi,canreinvite from sipusers where name='10000';
+-------+------------+-----------+-------+---------------+----------+------------+--------------+--------------+-------------+-------------+
| name  | defaultuser | host      | type  | context       | dtmfmode | insecure   | mailbox      | hasvoicemail | subscribemwi | canreinvite |
+-------+------------+-----------+-------+---------------+----------+------------+--------------+--------------+-------------+-------------+
| 10000 | 10000      | 10.0.1.207 | friend | pruebausuarios | rfc2833  | port,invite | 10000@default | yes          | no          | no          |
+-------+------------+-----------+-------+---------------+----------+------------+--------------+--------------+-------------+-------------+
1 row in set (0.00 sec)
```

Figura 9.32: Información Relevante de Tabla sipusers

  - name=10000: Nombre del usuario SIP.
  - host=10.0.1.207: Dominio o nombre de host del servidor SIP. Debido a que Kamailio es un SIP proxy con estado y a su vez es el que gestiona el Registro y Localización de usuarios, es requerido colocarlo a Kamailio como Dominio.
  - type=friend: Tipo de cliente. Este usuario está en la capacidad de realizar y recibir llamadas, por ende es un usuario del tipo friend.
  - context=pruebausuarios: Nombre del contexto al que pertenece el usuario. Muy recomendable en su uso con el Plan de Discado.
  - insecure=port,invite: Debido a que se desea que Asterisk actúe como un servidor de multimedia y se tiene a Kamailio con la capacidad de ofrecer de manera más efectiva servicios SIP, Kamailio es el que se encarga de autenticar las sesiones SIP, así pudiendo deshabilitar la autenticación SIP de Asterisk.
  - mailbox=10000@default: Es el buzón de voz correspondiente a ese usuario SIP en particular (el “10000” hace referencia al nombre del buzón de voz y “default” es el contexto del buzón, ambos definidos en la tabla de voicemail).
  - subscribemwi=no: El proceso de notificación de voicemails se basa en una sesión diferente a la de INVITE o REGISTER, esta se basa en SUBSCRIBE/NOTIFY. Debido a que un usuario se suscribe solo a un Asterisk y Asterisk no maneja mecanismos para mantener estas subscripciones compartidas (las mantiene en memoria RAM), se permite que los mensajes NOTIFY sean enviados así el usuario no esté suscrito.
- **voicemail**: En esta tabla es donde se mantiene la configuración de los buzones de voz de manera compartida por base de datos.

```
mysql> select context,mailbox,password from voicemail where mailbox="10000";
+---------+---------+----------+
| context | mailbox | password |
+---------+---------+----------+
| default | 10000   | 1234     |
+---------+---------+----------+
1 row in set (0.03 sec)
```

Figura 9.33: Información Relevante de la Tabla voicemail

  - context=default: El contexto en el cual se encuentra este buzón de voz (no tiene ninguna relación con los contextos manejados por los usuarios SIP).
  - mailbox=10000: Nombre del buzón de voz (la combinación de ambos permiten darle la información necesaria al usuario SIP para acceder al mismo, 10000@default).

- password=1234: Método de autenticación para validar el acceso al buzón de voz.

Archivos de Configuración

La configuración de Asterisk es dividido en múltiples archivos de configuración, los modificados son:

- **sip.conf:** Aquí se especifica la configuración del protocolo de señalización SIP, como sus canales y los usuarios SIP. Debido a que los usuarios son configurados en base de datos, solo algunas características son colocadas en este archivo para afectar a los usuarios de manera global. El resultado puede verse en la Figura 9.34.

```
10: ;#############  CONFIGURACION GLOBAL DE LOS CANALES SIP DE ASTERISK #########
11: [general]
12: context=default
13: transport=udp
14: bindport=5060
15: bindaddr=0.0.0.0
16: allowguest=no
17: srvlookup=yes
18: disallow=all
19: allow=alaw
20: allow=ulaw
21: allow=gsm
22: language=en
23: rtcachefriends=yes
24: ;#################################################################################
```

Figura 9.34: Configuración del Módulo de Canal SIP en Asterisk

  - transport, bindport, bindaddr: Utilizadas para la configuración de la apertura del socket por el que escuchará Asterisk.
  - disallow=all: Deshabilitar el uso de cualquier tipo de Codec.
  - allow=alaw,ulaw,gsm: Permitir el uso de los codecs listados, así solo se permite el uso de los codecs especificados en la aplicación de las llamadas.
  - rtcachefriend=yes: Debido a que se está utilizando Asterisk Realtime Arquitecture, Asterisk cargará de base de datos en memoria y la información del usuario cuando se accede al mismo. Esto habilita Voicemail MWI (mandar NOTIFY al suscribirse) y poder visualizar el estado del usuario con comandos como “sip show peers”.
- **rtp.conf:** Aquí se especifica la configuración del protocolo RTP. El resultado puede verse en la Figura 9.35.

```
10: [general]
11: ; RTP start and RTP end configure start and end addresses
12: rtpstart=35000
13: rtpend=55000
```

Figura 9.35: Configuración del Módulo RTP en Asterisk

  - rtpstart=35000, rtpend=55000: Rango de puertos RTP que Asterisk podría indicar en sus parámetros de RTP en la cabecera SDP de los mensajes SIP por el cual espera recepción de paquetes RTP por esa sesión.
- **extensions.conf:** Quizás el archivo más relevante de Asterisk. Este define el Plan de Discado de Asterisk, en pocas palabras, la lógica de enrutamiento principalmente de llamadas de los distintos mensajes que pueden llegar a Asterisk (en este caso mensajes basados en el protocolo SIP).

Ahora, la construcción del Dialplan consiste principalmente en contextos donde es agrupado la lógica del plan de discado, dependiendo de lo especificado en la configuración de SIP (todos los usuarios SIP registrados fueron colocados en el contexto “pruebausuarios”).

Dentro de los contextos se definen distintos comandos, el más utilizado en este caso es “exten” e “include”

- exten: el comando exten define la lógica de enrutamiento del plan de discado, éste recibe 3 parámetros, que son:
  - extensión: este primer parámetro hace referencia a la extensión que es marcada al llegar algún mensaje (SIP en este caso).
  - prioridad: este parámetro indica la secuencia por el cual un mensaje es interpretado dentro de la extensión a la cual hace match.
  - comando: Aquí se especifica el comando a utilizar, el cual usualmente es una aplicación que forma parte o es cargado en Asterisk. Algunas de las aplicaciones más relevantes son: Dial, VoiceMail, Set. GoTo, etc.

La Figura 9.36 muestra la lógica detrás del plan de discado de la implementación para el contexto pruebausuarios. Básicamente esta lógica consiste en utilizar a la aplicación Dial para comunicarse con el usuario a la que la extensión marcada hace match, en caso de no contestar en 30 segundos (código NOANSWER) se entra en la aplicación VoiceMail para dejarle un mensaje de voz a ese usuario. En caso de estar ocupado, Asterisk responde al usuario que realiza la llamada con el tono de ocupado.

```
10: [pruebausuarios]
11:
12: exten => _1XXXX,1,NoOp(Entrando por pruebausuarios, con Kamailio)
13: exten => _1XXXX,n,Set(EXTENSION=${EXTEN})
14: exten => _1XXXX,n,Goto(${EXTEN},4)
15: exten => _1XXXX,4,Dial(SIP/${EXTEN},30)
16: exten => _1XXXX,n,NoOp( Dial Status: ${DIALSTATUS})
17: exten => _1XXXX,n,Goto(s-${DIALSTATUS},1)
18:
19: exten => _2XXXX,1,NoOp(Entrando por pruebausuarios, solo Asterisk)
20: exten => _2XXXX,n,Set(EXTENSION=${EXTEN})
21: exten => _2XXXX,n,Goto(${EXTEN},4)
22: exten => _2XXXX,4,Dial(SIP/${EXTEN},30)
23: exten => _2XXXX,n,NoOp( Dial Status: ${DIALSTATUS})
24: exten => _2XXXX,n,Goto(s-${DIALSTATUS},1)
25:
26: exten => s-NOANSWER,1,VoiceMail(${EXTENSION}@default)
27: exten => s-BUSY,1,Busy()
28: exten => s-CANCEL,1,Hangup()
29: include=otros
```

Figura 9.36: Configuración de Plan de Discado para el Contexto pruebausuarios

Ahora, un contexto y lógica de enrutamiento aparte es utilizada para el mecanismo de monitoreo personalizado del RA de Asterisk, la misma puede verse en la Figura 9.37.

```
10: [haasterisk]
11:
12: ;HA-Asterisk Monitoring
13: exten => 501,1,Answer()
14: exten => 501,n,Set(TIMEOUT(digit)=1)
15: exten => 501,n,Wait(0.5)
16:
17: exten => 501,n,Background(hello-world)
18: exten => 501,n,WaitExten(5)
19:
20: exten => 2,1,GoTo(201,2)
21:
22: exten => 201,1,Answer()
23: exten => 201,2,Playback(hello-world)
24: exten => 201,3,Hangup()
25:
26: exten => t,1,Playback(goodbye)
27: exten => t,n,Hangup()
```

Figura 9.37: Configuración de Plan de Discado para el Monitoreo del RA de Asterisk

Con esto se culmina la Sección 9.4.4 que, al momento de terminar la configuración de los componentes Proxy/Registrar, se considera la arquitectura de telefonía de VoIP completa para proceder con la verificación y pruebas a dicha arquitectura para probar su desempeño.

9.4.5 Configuración de Componentes Proxy/Registrar

En este punto, se tocan todas las configuraciones por las cuales pasan ambas instancias para lograr lo especificado en los capítulos anteriores, algunos de los aspectos a tocar en este punto son:

- Configuración de RTPProxy
- Creación del Agente de Recurso OCF de RTPProxy
- Personalización del Agente de Recurso OCF de Kamailio
- Creación de Esquema de Clustering con componentes Proxy/Registrar
- Configuración General de Kamailio

9.4.5.1 Configuración de RTPProxy

En la Figura 9.38, se definen los parámetros de inicio del servicio de RTPproxy.

```
10: #RTPProxy params
11: OPTIONS="-l 10.0.1.206 -A 10.0.5.203 -s udp:127.0.0.1:7722 -r
    /var/lib/rtpproxy/sessions -L 65535 -d DBUG:LOG_LOCAL1 -m 35000 -M 55000 -F"
```

Figura 9.38: Parámetros de Ejecución del Servicio de RTPProxy

Los significados de los parámetros especificados en la Figura 2.38 son:

- -l 10.0.1.206: Dirección por la cual RTPproxy escucha paquetes RTP.
- -A 10.0.5.203: Dirección de advertencia. Opción nueva para permitir la utilización de RTPproxy detrás de NAT.

- -s udp:127.0.0.1:7722: Es un socket de control, el cual es utilizado por el módulo nathelper de Kamailio para comunicarse y gestionar los parámetros obtenidos mediante Kamailio
- -L 65535: Ajustar el número de conexiones abiertas persistentes (cantidad de Open File Descriptor admitidos).
- -m 35000 –M 55000: Rango de puertos RTP a utilizar que definirá RTPproxy para recibir paquetes RTP.

9.4.5.2 Creación del Agente de Recurso OCF de RTPProxy

A diferencia de los otros servicios que desean ser monitoreados por los esquemas de clustering, no se consiguió un agente de recurso OCF para la gestión del servicio de RTPProxy, las razones pueden ser un tanto claras, RTPProxy es un programa bastante sencillo sin capacidades internas de monitoreo del mismo y sin soluciones terceras para un monitoreo rápido aislado del servicio. Por ello, scripts de gestión de servicios como Systemd y LSB son suficientes para el monitoreo del estado de los procesos del servicio y usualmente vienen creados al integrarse como paquetes .rpm o .deb dependiendo de la distribución de Linux utilizada.

PaceMaker posee la capacidad de monitorear agentes de recursos basados en Systemd, el cual se contaba con uno al momento de instalar RTPProxy por un paquete .rpm. Pero, al momento de la práctica, se pudo observar un alto delay no esperado al aplicar mecanismos de failover. Se pudo observar que se debía a un delay de 2 segundos para el inicio/parado del servicio de RTPProxy que se encontraba siendo gestionado por PaceMaker mediante el recurso de Systemd.

Resulta que, desde el punto de vista de esquemas de clustering, los scripts de inicio basados por Systemd son considerados “broken” (incompletos), debido a que Systemd no termina enviando una señal de que el inicio/parado del servicio se haya completado de forma exitosa, sino que manda la señal de que se realizó el inicio/parado del servicio. Esto no es suficiente para gestores de recursos como PaceMaker, lo que implica a PaceMaker entrar en un proceso de polling cada 2 segundos para enviar una señal al agente de recurso de RTPProxy verificando si ya inició/paró el servicio correctamente, Systemd responde a la señal y en ese momento es que se considera activo.

Es por todo esto que se considera la creación de un agente de recurso OCF, pero, debido a que se puede contar con el script de inicio Systemd para la verificación del estado del proceso de RTPProxy, éste es utilizado internamente en el script de OCF para el inicio/monitoreo/parado del servicio. Aquí, el agente de recurso OCF funcionará como una capa intermedia para la construcción correcta y el envío de señales debidas a PaceMaker para una optimización en los tiempos de respuesta. Como un extra, se aprovecha el monitoreo a mayor granularidad del proceso y parado como contingencia en caso de que la unidad de Systemd falle y pueda perder referencia del proceso de RTPProxy anteriormente creado.

El recurso OCF consta de 3 tipos de operaciones, start, stop y monitor. Para la acción de start, se pasan por 3 procesos, los cuales serán descritos a continuación:

- Verificación de que no se encuentra ya iniciado

- Inicio del proceso
- Monitoreo de que haya iniciado correctamente

Los primeros dos procesos se pueden ver en la Figura 9.39.

```
10: rtpproxy_start() {
11:     local errorfile error output piddir
12:     ocf_log info "Starting RTPProxy"
13:     if
14:         rtpproxy_status
15:     then
16:         ocf_log info "rtpproxy is already running."
17:         return $OCF_SUCCESS
18:     fi
19:
20:     rtp_cmd="systemctl start rtpproxy.service"
21:
22:     ocf_log info "rtpproxy is already trying"
23:     errorfile=`mktemp`
24:     output=$(eval ${rtp_cmd} 2>>$errorfile)
25:     result=$?
26:     error=`cat $errorfile`
27:     rm -f $errorfile
```

Figura 9.39: Función de Inicio del Recurso de RTPProxy

En la Figura 9.39, La función rtpproxy_status (línea 14) se encarga de verificar que no exista otro proceso igual, como el que se va a iniciar, ya activo en el sistema (esta verificación se hace tanto con Systemd, como verificando manualmente por el proceso). En caso de existir, se envía una señal de que inició correctamente.

En caso de no existir, se inicia el servicio RTPProxy utilizando el script de inicio de Systemd (línea 20 y 24) y se almacena en una variable en resultado para imprimirlo en los logs en caso de falla (línea 26).

```
10: ocf_log info "rtpproxy is trying to start"
11:     if [ $result -eq 0 ]; then
12:         result=1
13:         while [ $result -ne 0 ]; do
14:             sleep 0.3
15:             rtpproxy_get_pid >/dev/null
16:             result=$?
17:         done
18:
19:         result=$OCF_ERR_GENERIC
20:         while [ $result -ne $OCF_SUCCESS ]; do
21:             sleep 0.3
22:             rtpproxy_monitor
23:             result=$?
24:             ocf_log info "UBICAR: monitor in start returned $result"
25:         done
26:     else
27:         ocf_log err "rtpproxy could not be started, $(rtpproxy_format_result
    $result "$output" "$error")"
28:         result=$OCF_ERR_GENERIC
29:     fi
30:
31:     return $result
32: }
```

Figura 9.40: Seguimiento de la Función de Inicio del Recurso RTPProxy

Una vez iniciado el servicio, se comprueba que el mismo haya iniciado de manera correcta. Para ello, se verifica, como lo indica la Figura 9.40, que el ID del proceso exista (línea 15), en caso de existir, luego se monitorea que el servicio este activo (línea 23). Todo esto se encuentra en un ciclo, ya que se intenta múltiples veces verificar que el servicio inició correctamente. En caso de iniciar correctamente, se envía la señal de que el servicio inició correctamente.

Para no hacer la explicación muy extensa, es importante saber que para la acción de monitor, se corre la función rtpproxy_monitor (la misma especificada dentro de la función reflejada en la Figura 9.39) en los intervalos especificados al configurar el recurso (ver Sección 2.4.5.3). En caso de la acción de stop, es un proceso parecido al de start, se intenta parar con Systemd y en caso de no haber culminado correctamente se intenta de manera forzada matando el proceso con Systemd y sino manualmente.

9.4.5.3 Personalización del Agente de Recurso OCF de Kamailio

A diferencia del recurso OCF de Asterisk, la personalización del agente de recurso de Kamailio es menor, en este caso se agregaron tres parámetros extras para permitir el

```
10:   case ${OCF_RESKEY_proto} in
11:     udp)  output=`$OCF_RESKEY_sipsak -s
    sip:$OCF_RESKEY_monitoring_ip:${OCF_RESKEY_port} -H $OCF_RESKEY_monitoring_ip --
    transport udp>/dev/null 2>>$errorfile`
12:           result=$?
13:         if [ $monitorip2 -eq 1 ]; then
14:           output=`$OCF_RESKEY_sipsak -s
    sip:$OCF_RESKEY_monitoring_ip2:${OCF_RESKEY_port} -H $OCF_RESKEY_monitoring_ip2
    --transport udp>/dev/null 2>>$errorfile`
15:           result2=$?
16:         fi
17:   …
18:   esac
19:     error=`cat $errorfile`
20:     rm -f $errorfile
21:     if [ $monitorip2 -ne 1 ]; then
22:         result2=0
23:     else
24:         if [ $result -ne 0 -o $result2 -ne 0 ]; then
25:         ocf_log info "Kamailio is running, but not functional as sipsak…"
26:         return $OCF_ERR_GENERIC
27:         fi
28:     fi
29:     return $OCF SUCCESS
```

monitoreo de más de una dirección IP mediante **SIPSAK v0.9.6** funcionando de manera correcta, debido a ciertos errores en el RA. Además, la posibilidad de agregar parámetros, para aumentar la memoria compartida y privada al crear la instancia de Kamailio. La Figura 9.41 muestra la lógica del proceso de monitoreo mediante SIPSAK del recurso de Kamailio.

Figura 9.41: Lógica de Monitoreo del Recurso de Kamailio mediante SIPSAK

Como se muestra en la Figura 9.41, se verifica, mediante mensajes OPTION, la respuesta de las distintas direcciones que se deseen monitorear (línea 17 y 20), en este caso los sockets de escucha en Kamailio que son tomados del archivo de configuración de Kamailio son las direcciones IP virtuales configuradas en el clustering.

Debido a que, en caso de tener que aplicar failover, esta misma dirección es asignada al otro nodo.

La Figura 9.42 muestra los parámetros -m y –M que permiten personalizar la cantidad de memoria compartida y privada respectivamente que tendrá la instancia de Kamailio al iniciar.

```
10: kam_cmd="${OCF_RESKEY_binary} -P ${OCF_RESKEY_pidfile} -f ${OCF_RESKEY_conffile}
    $listen_param -m ${OCF_RESKEY_shmem} -M ${OCF_RESKEY_pkg}"
```

Figura 9.9.42: Configuración de Comando a Ejecutar para el Inicio de Kamailio

Ya con la creación del recurso OCF de RTPProxy y la personalización del de Kamailio, se puede proceder a la creación del esquema de clustering de los componentes Proxy/Registrar.

9.4.5.4 Creación de Esquema de Clustering con Componentes Proxy/Registrar

A diferencia del cluster creado para Asterisk, aquí se desea crear un esquema de clustering Activo/Pasivo para los nodos que corren el servicios de Kamailio y RTPproxy. Pero antes de crear el esquema de clustering, se considera relevante mantener los archivos de configuración de manera compartida. Para ello, se configura la herramienta DRBD, la cual no es considerada en esquemas Activo/Activo como Asterisk debido a su ineficiencia al ser utilizado con sistemas de archivo de clustering como GFS2 o OCFS2 y limitación de crecimiento, pero en esquemas activo/pasivo es eficiente ofreciendo replicación de datos de manera rápida en particiones sin muchas operaciones de I/O (como es el caso común de archivos de configuración). Para la replicación de DRBD, primero se define en cada nodo el espacio a replicar en una partición, la partición creada se puede ver en la Figura 9.43.

```
Disk /dev/xvdb: 22.0 GB, 22011707392 bytes, 42991616 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk label type: dos
Disk identifier: 0x00083106

    Device Boot      Start         End      Blocks   Id  System
/dev/xvdb1            2048    41943039    20970496   83  Linux
/dev/xvdb2        41943040    42991615      524288   83  Linux
```

Figura 9.43: Partición para la Replicación con DRBD

La configuración de DRBD se basa en la creación de recursos DRBD. La configuración de dicho recurso en ambos nodos se puede ver en la Figura 9.44.

```
10: resource kamailioetc {
11: protocol C;
12: meta-disk internal;
13: device /dev/drbd1;
14:         net {
15:                 allow-two-primaries no;
16:                 …
17:         }
18:         disk {
19:                 on-io-error detach;
20:         }
21:         syncer {
22:                 verify-alg sha1;
23:         }
24:         on kam1 {
25:                 disk /dev/xvdb2;
26:                 address 10.0.3.204:7789;
27:         }
28:         on kam2 {
29:                 disk /dev/xvdb2;
30:                 address 10.0.3.205:7789;
31:         }
32: }
```

Figura 9.44: Configuración del Recurso de DRBD

- net { allow-two-primaries no }: Indica que se utilizará un esquema de Maestro/Esclavo (correspondiente para esquemas de clusters Activo/Pasivo).
- disk { on-io-error detach; }: Indica que, en caso de percibir un error en I/O de ese recurso, se desconectan los discos.
- syncer { verify-alg sha1; }: Utilizado para asegurar la integridad de la replicación mediante SHA1.
- on-kam1, on-kam2 { … }: Representa la partición a replicar y el socket por el cual el tráfico es replicado.

El recurso DRBD es llamado “kamailioetc” el cual define un nuevo dispositivo de bloque virtual (/dev/drbd1). La configuración del recurso permite luego crear la metadata del recurso, levantar el recurso creado y definir cuál recurso será el primario (uno de los nodos). Al final se formatea ese dispositivo en el recurso primario con el sistema de archivo que se desee (en este caso ext4). Una vez culminado, ambos recursos deben verse sincronizados, uno como primario y otro como secundario, conforme se puede ver en la Figura 9.45.

```
version: 8.4.7-1 (api:1/proto:86-101)
GIT-hash: 3a6a769340ef93b1ba2792c6461250790795db49 build by phil@Build64R7, 2016-01-12 14:29:40

 1: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
    ns:8212 nr:12 dw:41 dr:9271 al:2 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Figura 9.45: Resultado de Sincronización de Recursos mediante DRBD

Ahora, como en Asterisk, se desea crear un cluster, pero en este caso, Activo/Pasivo con Kamailio. Para ello se utiliza Pacemaker y Corosync, gestionados por PCSD. La creación inicial del cluster es la misma a la de Asterisk.

Nodos pertenecientes al esquema de clustering

```
Cluster Name: kamcluster
Corosync Nodes:
 kam1vs3 kam2vs3
Pacemaker Nodes:
 kam1vs3 kam2vs3
```

Figura 9.46: Nodos Pertenecientes al Esquema de Clustering de Componentes Proxy/Registrar

La Figura 9.46 resalta los siguientes aspectos:

- **Cluster Name:** kamcluster
- **Corosync Nodes:** kam1vs3 y kam2vs3 (Activo/Pasivo)
- **Pacemaker Nodes:** kam1vs3 y kam2vs3 (Activo/Pasivo)

Configuración de Recursos Monitoreados

La Figura 9.47 muestra la configuración de los distintos recursos del esquema de clustering:

```
Resources:
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: ip=10.0.1.206 cidr_netmask=32
  Operations: start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
              monitor interval=10s (ClusterIP-monitor-interval-10s)
 Resource: ClusterIP2 (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: ip=10.0.1.207 cidr_netmask=32
  Operations: start interval=0s timeout=20s (ClusterIP2-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP2-stop-interval-0s)
              monitor interval=10s (ClusterIP2-monitor-interval-10s)
 Resource: kamailioetcfs (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd1 directory=/etc/kamailio fstype=ext4
  Operations: start interval=0s timeout=60 (kamailioetcfs-start-interval-0s)
              monitor interval=10s on-fail=fence (kamailioetcfs-monitor-interval-10s)
              stop interval=0s on-fail=fence (kamailioetcfs-stop-interval-0s)
 Clone: fence_kam2_xvm-clone
  Meta Attrs: interleave=true clone-max=2 clone-node-max=1
  Resource: fence_kam2_xvm (class=stonith type=fence_xvm)
   Attributes: port=tegamjg_kam2 pcmk_host_list=kam2vs3
   Operations: monitor interval=60s (fence_kam2_xvm-monitor-interval-60s)
  Master: kamailioetcclone
   Meta Attrs: master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true on-fail=fence
   Resource: kamailioetc (class=ocf provider=linbit type=drbd)
    Attributes: drbd_resource=kamailioetc
    Operations: start interval=0s timeout=240 (kamailioetc-start-interval-0s)
                promote interval=0s on-fail=fence (kamailioetc-promote-interval-0s)
                demote interval=0s on-fail=fence (kamailioetc-demote-interval-0s)
                stop interval=0s on-fail=fence (kamailioetc-stop-interval-0s)
                monitor interval=10s (kamailioetc-monitor-interval-10s)
 Clone: fence_kam1_xvm-clone
  Meta Attrs: interleave=true clone-max=2 clone-node-max=1
  Resource: fence_kam1_xvm (class=stonith type=fence_xvm)
   Attributes: port=tegamjg_kam1 pcmk_host_list=kam1vs3
   Operations: monitor interval=60s (fence_kam1_xvm-monitor-interval-60s)
 Resource: kamailiocluster (class=ocf provider=heartbeat type=kamailio)
  Attributes: listen_address=10.0.1.206 conffile=/etc/kamailio/kamailio.cfg pidfile=/var/run/kamailio
.pid monitoring_ip=10.0.1.206 monitoring_ip2=10.0.1.207 port=5060 proto=udp kamctlrc=/etc/kamailio/ka
mctlrc shmem=128 pkg=8
  Operations: start interval=0s timeout=60 (kamailiocluster-start-interval-0s)
              stop interval=0s timeout=30 (kamailiocluster-stop-interval-0s)
              monitor interval=5s (kamailiocluster-monitor-interval-5s)
 Resource: rtpproxycluster (class=ocf provider=heartbeat type=rtpproxy)
  Operations: start interval=0s timeout=60 (rtpproxycluster-start-interval-0s)
              stop interval=0s timeout=30 (rtpproxycluster-stop-interval-0s)
              monitor interval=10s (rtpproxycluster-monitor-interval-10s)
```

Figura 9.47: Configuración de los Recursos para el Cluster de Componentes Proxy/Registrar

En el esquema de clustering de componentes Proxy/Registra, se definieron 8 recursos, 2 de ellos son dispositivos STONITH (fencing). Los detalles de cada recurso se describen a continuación:

- **ClusterIP:** Recurso creado para crear una IP virtual nueva, como iniciar, monitorear y parar el estado de dicha dirección. El uso de IP virtuales permiten, que en caso de migración de recursos a otro nodo, el acceso al servicio sea transparente para cualquiera
  - ip=10.0.1.206: Dirección de IP virtual creada para asignar al servidor.
  - cidr_netmask=32: Como es una única dirección fija, se coloca mascara 32.
  - monitor: monitoreo cada 10 segundos (interval=10s).
- **ClusterIP2:** Recurso creado para crear una IP virtual nueva, como iniciar, monitorear y parar el estado de dicha dirección.
  - ip=10.0.1.207: Dirección de IP virtual creada para asignar al servidor.
  - cidr_netmask=32: Como es una única dirección fija, se coloca máscara 32.
  - monitor: monitoreo cada 10 segundos (interval=10s).
- **rtpproxycluster:** Recurso creado para iniciar, monitorear y parar el componente de RTPproxy
  - monitor: monitoreo cada 10 segundos (interval=10s).
- **kamailioetcfs:** Recurso creado para iniciar (montar), monitorear y parar (desmontar) el sistema de archivo especificado.
  - device=/dev/drbd1: Dispositivo donde esta formateado la partición que se desea montar (dispositivo de bloque virtual creado por DRBD).
  - directory=/etc/kamailio: Ruta donde se desea montar dicha partición.
  - fstype=ext4: Tipo de sistema de archivo que se desea montar, en este caso ext4.
  - monitor: monitoreo cada 10 segundos (interval=10s), en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence).
  - stop: en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence).
- **kamailioetcclone:** Recurso creado para iniciar, monitorear y parar el servicio de DRBD. Este es un recurso clonado con multi-estados, como ya explicado anteriormente.
  - Meta Atributos
    - master-max=1: Este parámetro indica que solo una instancia del mismo recurso clonado puede estar en estado Maestro.
    - master-node-max=1: Este parámetro indica que solo una instancia del mismo recurso en un mismo nodo puede estar en estado Maestro.
    - clone-node=2: Explicado anteriormente.
    - clone-node-max=1: Explicado anteriormente.
  - monitor: monitoreo cada 10 segundos (interval=10s), en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence).
  - stop: en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence).
  - promote: en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence).
  - Promote es el proceso de modificación de un recurso de Esclavo a Maestro (Secundario a Primario).
  - demote: en caso de fallar debe aplicarse fencing en este nodo (on-fail=fence). Demote es el proceso de modificación de un recurso de Maestro a Esclavo (Primario a Secundario).
- **kamailiocluster:** Recurso creado para iniciar, monitorear y parar el servicio de Kamailio
  - listen_address=10.0.1.206: Dirección IP por la cual el servicio de Kamailio escucha. Sin embargo, este es modificado por el especificado en el archivo de configuración de Kamailio.

- conffile=/etc/kamailio/kamailio.cfg: Ruta en donde se encuentra el archivo de configuración principal de Kamailio.
- pidfile=/var/run/kamailio.pid: Ubicación donde se encuentra el archivo con la información del identificador de proceso de Kamailio.
- monitoring_ip=10.0.1.206 monitoring_ip2=10.0.1.207: Direcciones que desean ser monitoreadas mediante SIPSAK con mensajes OPTIONS (El segundo parámetro es personalizado).
- port=5060 proto=udp: Puerto y protocolo para definir el socket por el que escucha, sin embargo, este es modificado por lo especificado en el archivo de configuración de Kamailio.
- kamctlrc=/etc/kamailio/kamailioctlrc: Ruta en donde se encuentra el archivo de configuración de control de Kamailio. Este especifica cual motor de control se está usando y su ubicación, para verificar con más detalle el funcionamiento correcto de Kamailio mediante el comando kamctl.
- monitor: monitoreo cada 5 segundos (interval=5s).
- shmem=128: Especifica la cantidad de memoria compartida que tendrá la instancia de Kamailio al iniciar, parámetro personalizado (default: 64).
- pkg= 8: Especifica la cantidad de memoria privada que tendrá la instancia de Kamailio al iniciar, parámetro personalizado (default: 8).

- **fence_kam1_xvm:** Recurso creado para monitorear la necesidad de mandar una señal al hipervisor para aplicar el fencing a Kamailio 1.
  - port=tegamjg_kam1: Nombre de la máquina virtual a la cual aplicarle fencing.
  - pcmk_host_list=kam1vs3: Lista de máquinas controladas por este dispositivo de fencing.
- **fence_kam2_xvm:** Recurso creado para monitorear la necesidad de mandar una señal al hipervisor para aplicar el fencing a Kamailio 2.
  - port=tegamjg_kam2: Nombre de la máquina virtual a la cual aplicarle fencing.
  - pcmk_host_list=kam2vs3: Lista de máquinas controladas por este dispositivo de fencing.

Restricciones de los Recursos

La Figura 9.48 muestra la configuración de las restricciones de los distintos recursos que forman parte del esquema de clustering de componentes Proxy/Registrar:

```
Location Constraints:
Ordering Constraints:
  start fence_kam1_xvm-clone then start fence_kam2_xvm-clone (kind:Mandatory)
  start fence_kam2_xvm-clone then promote kamailioetcclone (kind:Mandatory)
  promote kamailioetcclone then start kamailioetcfs (kind:Optional)
  Resource Sets:
    set kamailioetcfs sequential=true set ClusterIP ClusterIP2 sequential=false
set rtpproxycluster kamailiocluster sequential=true
Colocation Constraints:
  ClusterIP2 with ClusterIP (score:INFINITY)
  ClusterIP with kamailioetcfs (score:INFINITY)
  kamailioetcfs with kamailioetcclone (score:INFINITY) (with-rsc-role:Master)
  fence_kam2_xvm-clone with fence_kam1_xvm-clone (score:INFINITY)
  kamailioetcclone with fence_kam2_xvm-clone (score:INFINITY)
  kamailiocluster with rtpproxycluster (score:INFINITY)
  rtpproxycluster with ClusterIP2 (score:INFINITY)
```

Figura 9.48: Configuración de Restricciones de los Recursos Gestionados

Ciertas restricciones son aplicadas en los recursos del cluster. Dos tipos de restricciones son utilizadas aquí:

- **Ordering Constraints:** Esta restricción es utilizada para aplicar el orden en que los recursos de un cluster deben iniciar. La Figura 9.49 describe la restricción de orden de este esquema.

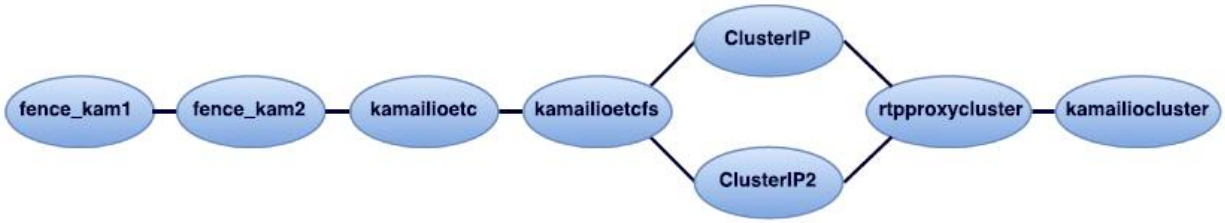


Figura 9.49: Restricción de Orden del Esquema de Clustering de Componentes Proxy/Registrar

Como lo muestra la Figura 9.48, la especificación de “Resource Sets” es utilizada para definir a los recursos ClusterIP y ClusterIP2 iniciar de manera no secuencial, pero una vez empezadas ambas, se prosigue con las restricciones.
Además de utilizar el tipo mandatorio (kind: Mandatory), también es utilizado el tipo opcional (kind: Optional), la diferencia es que el opcional es un poco menos restrictivo y es solo aplicado en acciones de inicio o paro de recursos, lo cual es un aspecto relevante cuando se tienen recursos clonados con multi-estados (como DRBD) que manejan otras acciones además de comienzo/paro de recursos (promote y demote, como comentado anteriormente). Esto evita un comportamiento inusual al momento de aplicar estas acciones para identificar al recurso Maestro y Esclavo.

- **Colocation Constraints:** Este tipo de restricción es utilizada para aplicar una restricción sobre los recursos para que un recurso no inicie en un nodo donde otro recurso no se encuentra. En este cluster, significa que todos los recursos deben estar iniciados en un solo nodo o sino en el otro. Muy útil en esquemas Activo/Pasivo.

Propiedades a nivel del Cluster

La Figura 9.50 muestra la configuración de las propiedades generales a nivel del esquema de clustering de componentes Proxy/Registrar:

```
Resources Defaults:
 migration-threshold: 2
 failure-timeout: 10m
 resource-stickiness: 200
Operations Defaults:
 No defaults set

Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: kamcluster
 dc-version: 1.1.13-10.el7_2.2-44eb2dd
 have-watchdog: false
 last-lrm-refresh: 1471640689
 no-quorum-policy: ignore
 start-failure-is-fatal: false
 stonith-action: reboot
 stonith-enabled: true
```

Figura 9.50: Propiedades a Nivel del Esquema de Clustering de Componentes Proxy/Registrar

Dos tipos de propiedades fueron especificadas, las cuales fueron explicadas anteriormente en la Sección 9.4.4.3.

9.4.5.5 Configuración de Kamailio

Primero, se configura el archivo "/etc/kamailio/kamctlrc". Este archivo se encarga de la configuración inicial para la creación de la base de datos para el almacenamiento de la información de Kamailio, como también el componente a utilizar para la gestión del mismo. FIFO es una interfaz de gestión y motor de control de la herramienta de Kamailio utilizando el comando "kamctl". Además, es utilizado por el agente de recursos OCF de Kamailio para monitorear a mayor granularidad el estado de Kamailio. El archivo se puede ver en la Figura 9.51.

```
10: # If you want to setup a database with kamdbctl, you must at least specify
11: # this parameter.
12: DBENGINE=MYSQL
13: ## database host
14: DBHOST=db1vs2
15: ## database name (for ORACLE this is TNS name)
16: DBNAME=kamailio_db
17: ## database read/write user
18: DBRWUSER="kamdb"
19: ## password for database read/write user
20: DBRWPW="!Sys64738_"
21: ## database read only user
22: DBROUSER="kamdb"
23: ## password for database read only user
24: DBROPW="!Sys64738_"
25: ## database access host (from where is kamctl used)
26: DBACCESSHOST=kam1vs2
27: …
28: ## - default FIFO
29: CTLENGINE="FIFO"
30: ## path to FIFO file
31: FIFOPATH="/tmp/kamailio_fifo"
```

Figura 9.51: Configuración Inicial de Kamailio y Métodos de Control

Base de Datos

Al momento de crear la base de datos con el comando kamdbctl, se crean muchas tablas, de las cuales, no todas tienen la necesidad de ser tomadas en cuenta. Sin embargo, hay algunas que son relevantes para la operación de la implementación, entre ellas están:

- **subscriber:** Tabla donde se definen los usuarios SIP del sistema.

```
mysql> select username,domain,password,ha1 from subscriber where username='10000';
+----------+------------+------------+----------------------------------+
| username | domain     | password   | ha1                              |
+----------+------------+------------+----------------------------------+
| 10000    | 10.0.1.207 | prueba0000 | 2736d20019bb3253e35212af0fccb3c2 |
+----------+------------+------------+----------------------------------+
1 row in set (0.00 sec)
```

Figura 9.52: Información Relevante de la Tabla subscriber

- username=10000: Nombre del usuario SIP.
- domain=10.0.1.207: Dominio el cual pertenece el usuario (para registrarse aquí debe ser el mismo de Kamailio).
- password=prueba0000: Contraseña con la cual debe autenticarse el usuario SIP.

- ha1=<hash>: Es el resultado de aplicar MD5 del username más el realm y el password (md5(username:realm:password)).

- **location**

```
mysql> select username,contact,received,expires,last_modified,cflags,socket from location;
+----------+------------------------------+----------+---------------------+---------------------+--------+---------------------+
| username | contact                      | received | expires             | last_modified       | cflags | socket              |
+----------+------------------------------+----------+---------------------+---------------------+--------+---------------------+
| 10001    | sip:10001@192.168.14.51:5063 | NULL     | 2016-08-24 22:26:56 | 2016-08-24 21:26:56 |      0 | udp:10.0.1.207:5060 |
+----------+------------------------------+----------+---------------------+---------------------+--------+---------------------+
1 row in set (0.00 sec)
```

Figura 9.53: Información Relevante de la Tabla location

- username: Usuario SIP que fue registrado.
- contact: Información de la cabecera del Contact de su mensaje SIP (esta suele ser suficiente para saber la localización del usuario).
- received: este viene como parámetro, que principalmente forma parte de la cabecera Via del mensaje SIP. Dicho parámetro es utilizado para saber de dónde proviene un mensaje SIP. Es útil cuando se cuenta con usuarios que se hallan en otra arquitectura de red donde no se encuentra gestionado el servidor SIP.
- cflags: Utilizado por Kamailio para indicar que esa entrada proviene o no de un usuario detrás de otra arquitectura de red.
- Socket: Socket por el cual fue recibido el mensaje REGISTER.

- **dispatcher:** Es la tabla asociada para la configuración de los mecanismos de balanceo de carga, especificando en ella aquellos servidores multimedia donde las sesiones SIP se encontrarán siendo balanceadas.

```
mysql> select * from dispatcher;
+----+-------+---------------------+-------+----------+---------------------------------+-------------+
| id | setid | destination         | flags | priority | attrs                           | description |
+----+-------+---------------------+-------+----------+---------------------------------+-------------+
|  1 |     0 | sip:10.0.1.200:5060 |     0 |        0 | duid=PBX1;my=pbx1;maxload=10000 | PBX1        |
|  2 |     0 | sip:10.0.1.201:5060 |     0 |        0 | duid=PBX2;my=pbx2;maxload=10000 | PBX2        |
+----+-------+---------------------+-------+----------+---------------------------------+-------------+
2 rows in set (0.00 sec)
```

Figura 9.54: Información Relevante de la Tabla dispatcher

- setid=0: Es el grupo al cual pertenece ese nodo, el módulo de dispatcher permite definir distintos grupos a los cuales una petición puede ser balanceada dependiendo de lo que sea necesario.
- destination: SIP URI del destino al cual las peticiones pueden ser balanceadas.
- priority=0: Es utilizado para colocar prioridad de ciertos nodos sobre otros, hay algoritmos que toman este valor en cuenta para sus mecanismos de balanceo.
- attrs: Ciertos atributos que pueden ser tomados en cuenta por el módulo dispatcher, entre ellos están:
  - duid: Es utilizado particularmente por el algoritmo de despacho por carga de llamadas (call load dispatcher) para identificar unívocamente, en el algoritmo, a un destino (nodo).
  - maxload: Utilizado también por el algoritmo de call load dispatcher para definir el máximo de carga activa que un nodo puede gestionar. En caso de cumplirse este valor, Kamailio se encargará de no enviarle más peticiones hasta que se cuente un valor menor de carga al especificado en el parámetro maxload.

Archivos de Configuración

A diferencia de Asterisk, Kamailio comprime toda su configuración en un mismo archivo llamado **Kamailio.cfg**. Kamailio divide su configuración en 3 diferentes secciones:

- **Definición de valores y parámetros globales:** Esta sección define los parámetros y valores globales como también los distintos flags a utilizar para facilitar la descripción, lógica y entendimiento del archivo de configuración.

```
10: ####### Defined Values #########
11: #
12: #### Defining Global Values ####
13: #
14: #!define WITH_MYSQL
15: #!define WITH_AUTH
16: #!define WITH_USRLOCDB
17: #!define WITH_ASTERISK
18: #!define WITH_LOADBALANCE
19: #!define WITH_NAT
20: #!define WITH_SIPSAK
21: #
22: ################################
23:
24: #!ifdef WITH_MYSQL
25: # - database URL - used to connect to database server by modules such
26: #       as: auth_db, acc, usrloc, a.s.o.
27: #!define DBURL "mysql://kamdb:!Sys64738_@db1vs2/kamailio_db"
28: #!ifdef WITH_ASTERISK
29: #!define DBASTURL "mysql://userdb:$Sys63738_@db1vs2/asterisk_db"
30: #!endif
```

```
10: # - flags
11: # FLT_ - per transaction (message) flags
12: # FLB_ - per branch flags
13: #!define FLT_ACC 1
14: #!define FLT_ACCMISSED 2
15: #!define FLT_ACCFAILED 3
16: #!define FLT_NATS 5
17: #!define FLB_NATB 6
18: #!define FLB_NATSIPPING 7
```

Figura 9.55: Valores Globales de la Configuración de Kamailio

Figura 9.56: Definición de Flags en la Configuración de Kamailio

- **flags:** La definición de flags como valores globales facilitan el entendimiento de la lógica de enrutamiento, además definen comportamientos particulares en la lógica de Kamailio. Dentro de los más relevantes mostradas en la Figura 9.56 están:
  - #!define FLB_NATSIPPING 7 (línea 18): Por ejemplo este flag es colocado a un usuario al ser registrado si se encuentra detrás de NAT, esto permitiría (como se ve más adelante en la definición del módulo) mantener la comunicación por parte de Kamailio y el endpoint.

```
10: ####### Global Parameters #########
11:
12: #!ifdef WITH_DEBUG
13: debug=4
14: log_stderror=no
15: #!else
16: debug=2
17: log_stderror=no
18: #!endif
19: log_facility=LOG_LOCAL0
20: fork=yes
21: children=4
22: # Changing default user agent header, it was causing problems for being too long
23: user_agent_header="User-Agent: TEGAMJG"
24: # Configuration of Socket listeners
25: #!ifdef WITH_NAT
26: listen=udp:10.0.1.207:5060
27: listen=udp:10.0.1.206:5060 advertise 10.0.5.203:5060
28: #!else
29: listen=udp:10.0.1.206:5060
30: #!endif
31: mpath="/usr/lib64/kamailio/modules/"
```

Figura 9.57: Parámetros Globales de la Configuración de Kamailio

- **Fork=yes, children=4 (Figura 9.57, línea 20, 21):** Es utilizado para crear 1 proceso por cada interfaz de red de escucha empleado y 4 procesos hijos por cada proceso. Esto puede permitir un procesamiento más rápido pero más costoso.
- **listen (Figura 9.57, línea 20, 21):** Define los sockets a crear, advertise es utilizado principalmente cuando se desea solucionar inconvenientes con usuarios detrás de NAT cuando Kamailio también se encuentra detrás de un NAT. Lo utiliza para definir sus parámetros de RTP en el mensaje SIP.

- **Sección de Módulos:** La sección de módulos podría dividirse en dos, en una parte se cargan los módulos que se desean utilizar y en la otra se definen los parámetros a utilizar por cada módulo. Los módulos presentados en la Figura 9.58 en la izquierda, son en su mayoría módulos básicos utilizados en la configuración original de Kamailio, los de la derecha son módulos nuevos considerados a configurar y adaptar la configuración original para ofrecer distintas funcionalidades como: la resolución de NAT Traversal, autenticación SIP, mecanismos de balanceo de carga, etc. Algunos de los módulos importantes son:

```
10: ####### Modules Section ########
11: loadmodule "mi_fifo.so"
12: loadmodule "kex.so"
13: loadmodule "tm.so"
14: loadmodule "tmx.so"
15: loadmodule "sl.so"
16: loadmodule "rr.so"
17: loadmodule "pv.so"
18: loadmodule "maxfwd.so"
19: loadmodule "usrloc.so"
20: loadmodule "registrar.so"
21: loadmodule "textops.so"
22: loadmodule "siputils.so"
23: loadmodule "xlog.so"
24: loadmodule "sanity.so"
25: loadmodule "ctl.so"
26: loadmodule "cfg_rpc.so"
27: loadmodule "mi_rpc.so"
28: loadmodule "acc.so"
29:
30:
31:
```

```
10: #!ifdef WITH_MYSQL
11: loadmodule "db_mysql.so"
12: #!endif.
13: #!ifdef WITH_AUTH
14: loadmodule "auth.so"
15: loadmodule "auth_db.so"
16: #!endif
17: #!ifdef WITH_PRESENCE
18: loadmodule "presence.so"
19: loadmodule "presence_xml.so"
20: loadmodule "presence_mwi.so"
21: #!endif
22: #!ifdef WITH_NAT
23: loadmodule "nathelper.so"
24: loadmodule "rtpproxy.so"
25: #!endif
26: #!ifdef WITH_DEBUG
27: loadmodule "debugger.so"
28: #!endif
29: #!ifdef WITH_LOADBALANCE
30: loadmodule "dispatcher.so"
31: #!endif
```

Figura 9.58: Módulos Cargados en la Configuración de Kamailio

- **usrloc.so:** Módulo que ofrece las funcionalidades de localización de usuarios SIP.
- **registrar.so:** Módulo que ofrece las funcionalidades para convertir a Kamailio en un registrar server.
- **auth.so y auth_db.so:** Módulos que ofrecen, en combinación, la funcionalidad de autenticación por parte de Kamailio utilizando una base de datos.
- **nathelper.so y rtpproxy.so:** Módulos que ofrecen, en combinación, la funcionalidad para gestionar usuarios y llamadas detrás de otras NATs.
- **dispatcher.so:** Módulo que ofrece la funcionalidad de balanceo de carga.

De aquí en adelante, se definen los parámetros configurados más relevantes de los distintos módulos a utilizar en la implementación, entre ellos están:

```
10: # ----------------- setting module-specific parameters ---------------
11: …
12: # ----- tm params -----
13: # auto-discard branches from previous serial forking leg
14: modparam("tm", "failure_reply_mode", 3)
15: # default retransmission timeout: 30sec
16: modparam("tm", "fr_timer", 30000)
17: # invite retransmission timeout for final reply
18: modparam("tm", "fr_inv_timer", 60000)
19: # ----------------------
```

Figura 9.59: Parámetros del Módulo TM en la Configuración de Kamailio

- **modparam("tm", "fr_inv_timer", 60000) (Figura 9.59, línea 45):** Este timer es utilizado para definir el timeout en milisegundos del envío de una petición INVITE sin recibir una respuesta final (Cualquier código de respuesta que no sea 1xx). Esto es utilizado para habilitar ciertos mecanismos de failover como será explicado en el Capítulo 10.5.1

```
10: # ----- auth_db params -----
11: #!ifdef WITH_AUTH
12: modparam("auth_db", "calculate_ha1", 1)
13: modparam("auth_db", "load_credentials", "")
14: modparam("auth_db", "db_url", DBURL)
15: modparam("auth_db", "password_column", "password")
16: modparam("auth_db", "use_domain", MULTIDOMAIN)
17: #!endif
18: # --------------------------
```

Figura 9.60: Parámetros del Módulo auth_db en la Configuración de Kamailio

- **modparam("auth_db","calculate_ha1", 1) (Figura 9.60, línea 12):** Parámetro que define que el valor hash definido en la columna ha1 de la tabla subscriber es calculado de la columna de password.

```
10: # #!ifdef WITH_NAT
11: # ----- rtpproxy params -----
12: modparam("rtpproxy", "rtpproxy_sock", "udp:127.0.0.1:7722")
13: # --------------------------
14:
15: # ----- nathelper params -----
16: modparam("nathelper", "natping_interval", 30)
17: modparam("nathelper", "ping_nated_only", 1)
18: modparam("nathelper", "sipping_bflag", FLB_NATSIPPING)
19: modparam("nathelper", "sipping_from", "sip:kamailio@10.0.1.206")
20: # params needed for NAT traversal in other modules
21: modparam("nathelper|registrar", "received_avp", "$avp(RECEIVED)")
22: modparam("usrloc", "nat_bflag", FLB_NATB)
23: # --------------------------
24: #!endif
```

Figura 9.61: Parámetros del Módulo nathelper y rtpproxy en la Configuración de Kamailio

- **modparam("rtpproxy","rtpproxy_sock","udp:127.0.0.1:7722") (Figura 9.61, línea 12):** Parámetro que define el socket a utilizar para el control entre rtpproxy y Kamailio (éste debe coincidir con el especificado en el parámetro de inicio del servicio de RTPproxy.
- **modparam("nathelper","natping_interval"-"ping_nated_only"-"sipping_bflag"-"sippingfrom") (Figura 9.61, línea 16 a la 20):** Estos parámetros son utilizados para definir el mecanismo para mantener conexiones abiertas en usuarios que están detrás de otras NATs. Esto consiste en mandar mensajes OPTION a esos usuarios y recibiendo respuestas de esos mensajes para asegurar el mantenimiento de la conexión en la tabla NAT.

```
10: #!ifdef WITH_LOADBALANCE
11: # ------ dispatcher params ------
12: modparam("dispatcher", "db_url", DBURL)
13: modparam("dispatcher", "table_name", "dispatcher")
14: modparam("dispatcher", "flags", 2)
15: modparam("dispatcher", "dst_avp", "$avp(dsdst)")
16: modparam("dispatcher", "grp_avp", "$avp(dsgrp)")
17: modparam("dispatcher", "cnt_avp", "$avp(dscnt)")
18: modparam("dispatcher", "dstid_avp", "$avp(dsdstid)")
19: modparam("dispatcher", "attrs_avp", "$avp(dsattrs)")
20: modparam("dispatcher", "attrs_pvname", "$var(attrs)")
21: modparam("dispatcher", "ds_hash_size",20)
22: modparam("dispatcher", "ds_hash_expire", 3600)
23: modparam("dispatcher", "ds_hash_initexpire", 60)
24: modparam("dispatcher", "ds_hash_check_interval", 30)
25: modparam("dispatcher", "ds_default_socket", "udp:10.0.1.207:5060")
26: modparam("dispatcher", "ds_ping_interval", 5)
27: modparam("dispatcher", "ds_ping_from", "sip:sipsrv@10.0.1.207")
28: modparam("dispatcher", "ds_probing_mode", 1)
29: modparam("dispatcher", "ds_probing_threshold", 1)
30: modparam("dispatcher", "ds_ping_reply_codes", "class=2;code=480;code=404")
31: # --------------------------------
32: #!endif
```

Figura 9.62: Parámetros del Módulo dispatcher en la Configuración de Kamailio

- **modparam(“dispatcher”,”flags”, 2) (Figura 9.62, línea 14):** Este parámetro indica al módulo de dispatcher que se va a hacer uso de mecanismos de failover, con los resultados que se comentarán en el capítulo 2.5.1. Para su correcto funcionamiento, requiere de ciertas condiciones, definir los AVP (como los especificados en la imagen anterior) y que Kamailio mantenga el estado de la transacción (mediante el módulo tm) de dicha transacción para que el mismo pueda reiniciar la transacción en otro nodo en caso de requerirlo
- **modparam(“dispatcher”,”avps”) (Figura 9.62, línea 15 a la 19):** Estos parámetros definen los AVPs para mantener, en tiempo real, información relacionada a los posibles nodos destinatarios de las peticiones, según el resultado del algoritmo de balanceo. Esto permite habilitar mecanismos de failover y otros tipos de algoritmo como call load dispatching
- **modparam(“dispatcher”,”ds_hash”) (Figura 9.62, línea 21 a la 24):** Estos parámetros definen las dimensiones de la tabla hash que debe gestionarse para el funcionamiento del algoritmo de call load dispatching, aunque es un algoritmo sin estado, requiere de mantener en una tabla un hash que contiene la combinación del call-id de la sesión y el duid del nodo de destino escogido
- **modparam(“dispatcher,”ds_ping”-“ds_probing”) (Figura 9.62, línea 26 a la 30):** Estos parámetros definen el mecanismo health check para conocer la operatividad de los nodos destinatarios que forman parte de la tabla de dispatcher. El mecanismo consiste en enviar mensajes OPTION cada cierto tiempo (línea 26), el resultado puede ser diferente a 200 (línea 30), ya que, por ejemplo, una respuesta 404 o 480 no significa una falla a nivel del servidor

- **Lógica de Enrutamiento**: De aquí en adelante, se configura toda la lógica de enrutamiento a tomar en cuenta por cada petición/respuesta o cualquier otro evento que un mensaje del protocolo de señalización SIP pase por Kamailio. Esta lógica es dividida por el tipo de mensaje que se recibe, si es una petición, respuesta o alguna falla. Dentro de ellas se define toda la lógica por la cual tiene que pasar cada tipo de petición

mediante “routes”, routes son funciones definidas donde se aplica la manipulación de cabeceras SIP o incluso SDP

- **request_route:** Bloque que define el comportamiento que tendrá cada inicio de transacción SIP. Cada vez que un paquete que determina el inicio de una transacción SIP pasa por Kamailio, su comportamiento será definido según las distintas funciones especificadas en ese bloque. En la Figura 9.63 se puede ver la configuración del bloque request_route:

```
10: ######## Routing Logic ########
11: request_route{
12: route(REQINIT);
13: # NAT detection
14: route(NATDETECT);
15: # handle requests within SIP dialogs
16: route(WITHINDLG);
17:    …
18:    t_check_trans();
19: # authentication
20: route(AUTH);
21: # record routing for dialog forming requests
22: remove_hf("Route");
23: if (is_method("INVITE|SUBSCRIBE"))
24: record_route();
25:     …
26:    # handle registrations
27: route(REGISTRAR);
28:    # request with no Username in RURI
29:  if ( $rU==$null )
30:  {
31:  sl_send_reply("484","Address Incomplete");
32:   exit;
33:  }
34: # user location service
35: route(LOCATION);
36: # relay message to destination
37: route(RELAY);
```

Figura 9.63: Configuración del Bloque request_route en la Configuración de Kamailio

- **RELAY:** RELAY es una de las funciones principales utilizadas para definir el comportamiento final antes del envio del mensaje dentro de esa transaccion SIP. La configuración del mismo puede verse en la Figura 9.64
  - MANAGE_BRANCH: Este bloque conlleva a un route que aplica la gestion de NAT en caso de ser necesario, y al culminar (al no cumplir con ninguno de los otros dos casos), se envía el mensaje dentro de esa transaccion

```
route[RELAY] {
#!ifdef WITH_NAT
        if (check_route_param("nat=yes"))
                setbflag(FLB_NATB);
#!endif
        # enable additional event routes for forwarded requests
        # - serial forking, RTP relaying handling, a.s.o.
        if (is_method("INVITE|BYE|SUBSCRIBE")) {
                if(!t_is_set("branch_route")) t_on_branch("MANAGE_BRANCH");
        }
        if (is_method("INVITE|SUBSCRIBE")) {
                if(!t_is_set("onreply_route")) t_on_reply("MANAGE_REPLY");
        }
        if (is_method("INVITE")) {
                if(!t_is_set("failure_route")) t_on_failure("MANAGE_FAILURE");
        }
        if (!t_relay()) {
                sl_reply_error();
        }
        exit;
}
```

Figura 9.64: Configuración de la Función RELAY

- **WITHINDLG**: Este route es utilizado para definir el comportamiento de una petición de transacción que se encuentra dentro de un mismo dialog (verificando si contiene el parametro "to" definido). Este route suele utilizarse mayormente dentro de una sesión INVITE debido a sus múltiples transacciones en una misma sesión, como mensajes ACK, BYE, CANCEL, etc. La configuración del mismo puede verse en la Figura 9.65

  Este route se encarga de eliminar la carga de una llamada de la tabla hash en caso de recibir un BYE o un CANCEL como se puede ver en la línea 15 y 16, además se encarga de mecanismos de enrutamiento de mensajes ACK mediante la información relacionada en las cabeceras de Record-Routes (línea

```
# Handle requests within SIP dialogs
route[WITHINDLG] {
 if (has_totag()) {
#!ifdef WITH_LOADBALANCE
#deleting call load from hash table
if(is_method("BYE|CANCEL"))
ds_load_update();
#!endif
  if (loose_route()) {
   if ( is_method("ACK") ) {
route(NATMANAGE);
   }
    route(RELAY);
  } else {
            …
     sl_send_reply("404","Not here");
}
```

  18 a la 22).

Figura 9.65: Configuración de la Función WITHINDLG

- **REGISTRAR**: La configuración de este route se puede ver en la Figura 9.66, dicho route se encarga de definir el comportamiento al llegar una petición REGISTER a Kamailio. Como ya fue dicho anteriormente, los routes son

llamados dentro de un bloque (en este caso el definido como request_route), el request_route determina que el route[NATDETECT] se ejecuta antes que el de REGISTRAR. Por ende, de esa petición se puede saber si el flag FLT_NATS es colocado (indicativo de que se detectó al usuario detrás de otra NAT). En ese caso, al ser registrado al usuario se coloca en la tabla de localización con sus respectivos flags por su conocimiento que se encuentra detrás de otra NAT (FLT_NATB y FLB_NATSIPPING). Al momento de intentar registrar al usuario (línea 20), se puede ver el parámetro 0x04, este es utilizado para guardar un único Address of Record (AoR) por usuario.

```
10: # Handle SIP registrations
11: route[REGISTRAR] {
12: if (is_method("REGISTER")) {
13:   if(isflagset(FLT_NATS)) {
14:    # Mark user in that branch NAT aware
15:    setbflag(FLB_NATB);
16:    # Enable SIP NAT pinging for the user
17:    setbflag(FLB_NATSIPPING);
18:   }
19: #!ifdef WITH_ASTERISK
20: if (!save("location","0x04")) {
21: sl_reply_error();
22:                 exit;
23:         }
24: #!endif
25:   exit;
26: }
27: }
```

Figura 9.66: Configuración de la Función REGISTRAR

- **LOCATION:** Este route es el que determina la localización de un usuario SIP, el cual se puede ver en la Figura 9.67. Cuando una petición viene de afuera y no de los nodos Asterisk que forman parte de la tabla de dispatcher, no se es determinada su localización. Lo que ocurre es que directamente se evalúa a cuál de los nodos de Asterisk será enviada esta petición llamando al route[TOASTERISK] (línea 15) o route[TOASTERISK_NONINVITE] (línea 19) dependiendo de los tipos de petición como lo indica la imagen

  Ahora, en caso de que la petición SIP venga de los nodos de Asterisk, Kamailio se encarga de ubicar la localización del usuario para poder hacer transmitir la petición al destinatario esperado (línea 25). En caso de fallar la búsqueda se replica con código 404.

```
10: # USER location service
11: route[LOCATION] {
12: #!ifdef WITH_ASTERISK
13:   if(is_method("INVITE") && (!route(FROMASTERISK))) {
14:           # Set Asterisk INVITE destination
15: route(TOASTERISK);
16:           exit;
17:   }else if(is_method("SUBSCRIBE") && (!route(FROMASTERISK))) {
18:                           # Set Asterisk SUBSCRIBE destination
19:                           route(TOASTERISK_NONINVITE);
20:                           exit;
21:                 }
22: }
23: #!endif
24: $avp(oexten) = $rU;
25: if (!lookup("location")) {
26: $var(rc) = $rc;
27: switch ($var(rc)) {
28:  case -1:
29:  case -3:
30:   send_reply("404", "Not Found");
31:   exit;
32:  …
33:     }
34: }
35:    …
```

Figura 9.67: Configuración de la Función LOCATION

- **AUTH**: Este es el route que define el comportamiento para la autenticación de peticiones SIP, por ende es una de las primeros routes a ejecutar en el bloque request_route. Además, en este route se define las respuestas a los mensajes OPTION para el monitoreo de la herramienta mediante el RA Kamailio de Pacemaker. La configuración de esta función es mostrada en la Figura 9.68.

  Es importante definir a los nodos de Asterisk como confiables y no tener que pasar por un proceso de autenticación, por ello se verifica si la petición viene de los nodos de Asterisk definidos en la tabla dispatcher, en caso de serlo se culmina y se sale del route. En caso contrario, Kamailio obliga a esta petición a entrar en un proceso de desafío de petición/respuesta (MD5/Digest)

```
10: # # Authentication route
11: route[AUTH] {
12:     …
13: #!ifdef WITH_SIPSAK
14: # Enable replies to Kamailio RA monitoring
15: if(uri==myself && is_method("OPTIONS")) {
16: options_reply();
17: }
18: #!endif
19:
20: #!ifdef WITH_AUTH
21: #!ifdef WITH_ASTERISK
22:  if(route(FROMASTERISK))
23:   return;
24: #!endif
25:
26:  if (is_method("REGISTER") || from_uri==myself) {
27:   # authenticate requests
28:   if (!auth_check("$fd", "subscriber", "1")) {
29:    auth_challenge("$fd", "0");
30:    exit;
31:   }
32:   # user authenticated - remove auth header
33:   if(!is_method("REGISTER|PUBLISH")) {
34:    consume_credentials();
35: }
36:  }
37: #!endif
38:  return;
39: }
```

Figura 9.68: Configuración de la Función AUTH

- **NATDETECT**: Este route es uno de los primeros en llamar dentro del bloque request_route, este permite definir de entrada si una petición viene de un endpoint detrás de otra arquitectura de red. La configuración de esta función puede verse en la Figura 9.69

  Las pruebas para demostrar si se encuentra detrás de otra arquitectura es, primero, verificar que la dirección especificada en la cabecera Contact venga de una dirección privada (línea 13). Luego, si esto cumple, verificar si la dirección o puerto fuente hace match con la especificada en la cabecera Via (línea 14). En caso de no ser así alguna de ellas, se asume que el usuario se encuentra detrás de otra arquitectura. En caso de determinar que se encuentra en otro NAT, ocurren dos cosas:
  - En caso de ser una petición REGISTER, este agrega un parámetro “received” incluyendo la dirección fuente y puerto por el cual se está recibiendo la petición en la cabecera Via (línea 18) y se agrega el flag FLT_NATS para ser interpretado al momento de registrar (línea 22)
  - En caso de ser una petición que no sea REGISTER, se reescribe la dirección fuente y puerto de la cabecera Contact (línea 20), esto tiene un impacto en la manera a donde se dirigen algunas de las próximos mensajes dentro de la misma sesión

```
10: # Caller NAT detection route
11: route[NATDETECT] {
12: #!ifdef WITH_NAT
13:  if (nat_uac_test("1")) {
14:  if (nat_uac_test("18")) {
15: force_rport();
16:  if (is_method("REGISTER")) {
17:    fix_nated_register();
18:   } else {
19: fix_nated_contact();
20:   }
21:   setflag(FLT_NATS);
22:  return;
23:  }
24:  }
25: #!endif
26:  return;
27: }
```

Figura 9.69: Configuración de la Función NATDETECT

- **NATMANAGE**: Este route es llamado dentro de algunos routes definidos en route[RELAY] e incluso en el bloque onreply_route, la función de este route es la de la modificación de cabeceras SIP e incluso SDP para definir el comportamiento del flujo de mensajes SIP y RTP donde alguno de los endpoints se encuentra detrás de otra arquitectura de red. La configuración de esta función puede verse en la Figura 9.70

  La lógica completa definida en este route es algo complejo. Si el flag FLT_NATS está puesto (obtenido del route[NATDETECT] previamente) y FLB_NATB no lo está, significa que pasará por la gestión del proxy RTP (línea 14). La función rtpproxy_manage es la que se encarga de comunicarse con el proxy RTP para la apertura de sesiones con el proxy RTP y permitir que haga relay del tráfico RTP basándose en los parámetros que se estén definiendo para el cuerpo SDP. Debido a que se está usando el proxy RTP detrás de NAT y no afuera asumiendo la configuración como un multihomed proxy RTP, la colocación de estos parámetros es forzada.

  Si el mensaje no viene desde los nodos Asterisk especificados en la tabla dispatcher, se fuerza que el envío sea por la dirección 10.0.1.207 (línea 22) y el parámetro en rtpproxy_manage se coloca la dirección 10.0.1.206 (línea 23). Lo anterior especifica que se modifica el parámetro de recepción multimedia en el cuerpo SDP a 10.0.1.206, lo que implica que cuando sea enviado al nodo Asterisk, el nodo debe interpretar que el trafico RTP será enviado al proxy RTP a la dirección 10.0.1.206, y luego el proxy RTP es el que se encarga de gestionar el relay hacia el endpoint correspondiente cuando el trafico RTP pase por el proxy RTP.

Ahora, si el mensaje viene desde los nodos Asterisk especificados en la tabla dispatcher, se fuerza el envío a la dirección 10.0.1.206 (línea 25) que es la que se tiene configurado NAT 1:1 con el firewall dentro de la implementación, y se

```
10: # RTPProxy control
11: route[NATMANAGE] {
12: #!ifdef WITH_NAT
13:     …
14:  if (!(isflagset(FLT_NATS) || isbflagset(FLB_NATB)))
15:    return;
16:  if(isbflagset(FLB_NATB) && !isflagset(FLT_NATS)) {
17:  if(is_request() && !has_totag() && !route(FROMASTERISK)) {
18:  return;
19:  }else if(route(FROMASTERISK) && is_reply()) return;
20:  }
21: if(!route(FROMASTERISK)){
22: force_send_socket(10.0.1.207);
23: rtpproxy_manage("rw","10.0.1.206");
24:  }else{
25: force_send_socket(10.0.1.206);
26:  rtpproxy_manage("rw","10.0.5.203");
27:  }
28: …
29:  return;
30: }
```

envía el mensaje al endpoint detrás de otra NAT con el parámetro de recepción multimedia en el cuerpo SDP a 10.0.5.203 (línea 26). Esto significa que cuando sea enviado al endpoint, este debe interpretar que el trafico RTP será enviado al proxy RTP a la dirección 10.0.5.203 y el proxy RTP luego es el que se encarga de gestionar el relay hacia el nodo Asterisk correspondiente cuando el trafico RTP pase por él.

Figura 9.70: Configuración de la Función NATMANAGE

- **MANAGE_REPLY:** Este route se encuentra dentro de un bloque onreply_route, esto significa que es la lógica especificada cuando se recibe un reply de algún request perteneciente a una transacción. La configuración de esta función puede verse en la Figura 9.71. La lógica interna se basa en aumentar la carga en la tabla de hash del algoritmo call load dispatching en caso de la recepción de un mensaje 2xx recibido por Asterisk (línea 14 y 15). Esto especifica que ese nodo de Asterisk tiene una carga más a tomar en cuenta por el algoritmo de call load dispatching para peticiones INVITE sucesivas

```
10: # Managing reply messages
11: onreply_route[MANAGE_REPLY] {
12: #!ifdef WITH_LOADBALANCE
13: if(is_method("INVITE") && ds_is_from_list("0")) {
14: if(status=~"2[0-9][0-9]") {
15: ds_load_update();
16: }else if(status=~"[3-7][0-9][0-9]") {
17: ds_load_unset();
18: }
19: }
20: #!endif
21: if(status=~"[12][0-9][0-9]"){
22:  route(NATMANAGE);
23:  }
24: }
```

Figura 9.71: Configuración de la Función MANAGE_REPLY

- **MANAGE_FAILURE:** Este route se encuentra dentro de un bloque failure_route, este es utilizado cuando ocurre una falla en una transacción activa dentro de Kamailio. La configuración de esta función puede verse en la Figura 9.72. Su lógica es principalmente utilizada para ofrecer los mecanismos de failover por la recepción de una respuesta con código 500 (indicando falla a nivel del servidor), cumplimiento del timeout especificado para la recepción de una respuesta o la no respuesta del nodo de Asterisk (línea 15). En caso de haber falla se marca como inactivo y se prueba con otro nodo (línea 17).

  Es importante destacar que es posible lograr esto debido a que se tienen a los nodos disponibles dentro de los AVPs mencionados en la definición de los módulos, y que los nodos destinatarios están ordenados en el AVP que lista los nodos destinatarios en orden de capacidad (debido a que se está utilizando el algoritmo call load dispatching, se ordenan por los que tienen menor carga de llamadas).

  En caso de haber otro posible destinatario en el AVP, se vuelve a enviar el mensaje pero al otro nodo Asterisk, colocando un timeout transaccional hasta obtener el primer reply de 4 segundos (línea 19), en caso de no recibir el primer reply se repite el proceso (línea 20).

```
10: Managing failed messages load balanced to Asterisk
11: failure_route[MANAGE_FAILURE] {
12:     …
13: #!ifdef WITH_LOADBALANCE
14:         if (t_check_status("500") || t_branch_timeout() || !t_branch_replied()){
15:                 ds_mark_dst("ip");
16:                 if(ds_next_dst()) {
17:                     t_set_fr(0,4000);
18:                     t_on_failure("MANAGE_FAILURE");
19:                     xlog("L_INFO","Dispatching to Asterisk Box $du\n");
20:                     route(RELAY);
21:                     exit;
22:                 }else{
23:                         send_reply("404", "No destination");
24:                         exit;
25:                 }
26: }
27: #!endif
28: }
```

Figura 9.72: Configuración de la Función MANAGE_FAILURE

- **MANAGE_FAILURE_NONINVITE:** La lógica de este route es la misma que la de MANAGE_FAILURE, se diferencia en que no es para peticiones INVITE y por ende el algoritmo utilizado es round-robin, en este caso es para peticiones SUBSCRIBE
- **FROMASTERISK:** La configuración de este route puede verse en la Figura 9.73. Dicho route es utilizado frecuentemente en la lógica del archivo de configuración y se encarga es de verificar si el trafico SIP viene de los nodos Asterisk de la tabla dispatcher (línea 14)
- **TOASTERISK**: La configuración de este route puede verse en la Figura 9.73. Dicho route es uno de los más importantes del archivo de configuración y es el que se encarga del mecanismo de balanceo de carga del tráfico SIP INVITE a los nodos Asterisk pertenecientes a la tabla dispatcher aplicando el algoritmo de call load dispatching al grupo de nodos Asterisk “0” (línea 25)

Al intentar escoger algún nodo y haber fallado en el intento, se envía un reply con código 500 indicando falla (línea 25). En cambio si se aplica el algoritmo y se logra escoger el nodo con menor carga, se coloca un timeout hasta obtener el primer reply antes de cumplir 4 segundos (línea 29), y en caso de fallar, esa transacción se envía a la ruta MANAGE_FAILURE anteriormente mencionada y se envía el trafico al nodo Asterisk escogido

```
10: #!ifdef WITH_ASTERISK
11: # Test if coming from Asterisk
12: route[FROMASTERISK] {
13: #!ifdef WITH_LOADBALANCE
14: if(ds_is_from_list("0")) {
15: return 1;
16: }else{
17: return -1;
18: }
19: #!endif
20: }
21:
22: # Send invite request to Asterisk
23: route[TOASTERISK] {
24: #!ifdef WITH_LOADBALANCE
25: if(!ds_select_dst("0", "10")) {
26:                 sl_send_reply("500", "Service Unavailable");
27:                 exit;
28:         }
29: t_set_fr(0,4000);
30: t_on_failure("MANAGE_FAILURE");
31: route(RELAY);
32: exit;
33: #!endif
34: }
```

Figura 9.73: Configuración de la Función FROMASTERISK y TOASTERISK

- **TOASTERISK_NONINVITE**: La lógica de este route es la misma que la de route[TOASTERISK] pero la diferencia es que se utiliza el algoritmo round-robin para peticiones que no son INVITE (en este caso SUBSCRIBE)

9.4.6 Configuración de Escenarios de Prueba con SIPp

Conviene experimentar la capacidad en la cual la implementación (con sus limitaciones a nivel de recursos) opera en ambientes de alta concurrencia de llamadas y registro de usuarios. Por ello, se determina el uso de SIPp para la creación de pruebas de alta concurrencia de realización de llamadas y registro de usuarios, para así comprobar el funcionamiento y la escalabilidad de la implementación.

Existen dos tipos de escenarios de prueba a realizar:

- **Flujo de sesiones INVITE:** Para el flujo de sesiones INVITE se crearon dos escenarios distintos
  - Un escenario que se encargue de realizar llamadas a otro endpoint que se encuentre corriendo SIPp también con intercambio de flujo RTP entre ellos
  - Un escenario que se encargue de realizar llamadas a un IVR dentro de Asterisk

- **Flujo de sesiones REGISTER:** Este escenario se encarga de mandar múltiples mensajes REGISTER para el registro de múltiples usuarios SIP. El rango de usuario es de 10000 hasta 20000, lo que significa una cantidad de 10000 usuarios SIP

Cada uno de los escenarios está definido por 3 archivos:

- /root/scripts/<nombre de archivo>.sh: En este archivo se definen los múltiples parámetros que son pasados al binario de SIPp para arrancar el escenario
- /usr/local/src/sipp-3.4.1/csv/<nombre de archivo>.csv: En este archivo se definen ciertos parámetros que son interpretados por el archivo encargado de determinar el escenario. Algunos de estos parámetros son: extensión a la que se marca, SIP URI, etc)
- /usr/local/src/sipp-3.4.1/scenarios/<nombre de archivo>.xml: En este archivo se define el escenario y flujo como tal de la prueba a realizar, además se define la cabecera SIP en bloques CDATA

Algunos de los escenarios consisten en lo siguiente:

9.4.6.1 Flujo de Mensajes INVITE entre dos Endpoints SIP

En este escenario se utilizan dos instancias diferentes de SIPp, simulándolos como endpoints para la realización de las llamadas. Es importante destacar que hay dos escenarios creados que tienen el mismo objetivo pero en distintas localizaciones, uno se encarga de simular al endpoint estando en la arquitectura de red de la implementación y el otro escenario se encarga de simular al endpoint detrás de otra arquitectura de red. El escenario consiste de:

- **Bash Script:** El próximo script define los parámetros que serán pasados al binario de SIPp para iniciar el flujo representado por el escenario, algunos de estos se muestran en la Figura 9.74:
  - CSV (línea 12): Ruta donde se encuentra el archivo CSV definido para el escenario a probar
  - RATE (línea 13): Este consiste en la tasa de llamadas a realizar por segundo
  - LIMIT (línea 14): Representa el límite de llamadas que se pueden encontrar activas en el flujo
  - SCENARIO (línea 18): Ruta donde se encuentra el archivo XML que define el escenario
  - UDPSOCK (línea 24):
    - u1: UDP mono socket, se utiliza un único socket para todo el flujo
    - un: UDP multi socket, se utiliza un socket por cada llamada dentro del flujo

```
10: #!/bin/sh
11: # Definiendo variables
12: CSV="/usr/local/src/sipp-3.4.1/csv/prueba_invite_nonat.csv"
13: RATE=$1
14: LIMIT=$2
15: CALLS=$3
16: SOURCE="10.0.1.208"
17: MEDIASOURCE="10.0.1.208"
18: SCENARIO="/usr/local/src/sipp-3.4.1/scenarios/prueba_invite_nonat.xml"
19: DESTINATION="10.0.1.207"
20: ERRFILE="/usr/local/src/sipp-3.4.1/scenarios/prueba_invite_nonat_err.log"
21: DBGFILE="/usr/local/src/sipp-3.4.1/scenarios/prueba_invite_nonat_calldebug.log"
22: MINRTP=35000
23: MAXRTP=55000
24: UDPSOCK=un
25: MAXSOCK=$(($2+10))
26:
27: # Ejecucion
28: cd /usr/local/src/sipp-3.4.1/
29: ./sipp -inf $CSV -r $RATE -i $SOURCE -m $CALLS -sf $SCENARIO -l $LIMIT
    $DESTINATION -mp $MINRTP -min_rtp_port $MINRTP -max_rtp_port $MAXRTP -t $UDPSOCK
    -max_socket $MAXSOCK -trace_err -error_file $ERRFILE
```

Figura 9.74: Definición de Bash Script que Ejecuta el Binario de SIPp

- **Archivo CSV:** El archivo CSV contiene parámetros que son utilizados en el escenario XML representados con [fieldX] donde X es un número entero indicando la columna. El archivo CSV se puede ver en la Figura 9.75:
  - SEQUENTIAL (línea 10): Esta fila indica que se utilizará de manera secuencial las filas representadas en el CSV por cada llamada, en caso de no haber más filas el proceso se repite desde el principio del archivo
  - Filas por llamadas (línea 11): Cada fila contiene múltiples columnas que representan los campos que serán utilizados en el archivo XML que define el escenario
    - 10000: Este define el usuario del endpoint que inicia la petición
    - 10.0.1.207: Este define a la parte de host que define el SIP URI
    - [authentication username=10000 password=prueba0000]: Este define los parámetros requeridos en caso de requerir pasar por el desafío de autenticación MD5/Digest
    - 15000: Extensión a la cual se está marcando

  Cada una de estas columnas son mapeadas en el archivo XML mediante la expresión [fieldX] donde X es un entero que comienza en 0 hasta la cantidad de columnas por línea, por ejemplo: [field0]=10000, [field1]=10.0.1.207.

```
10: SEQUENTIAL
11: 10000;10.0.1.207;[authentication username=10000 password=prueba0000];15000
```

Figura 9.75: Definición de Archivo CSV para SIPp

- **Archivo XML**: El archivo XML es el que define el escenario, la definición del archivo puede verse en la Figura 9.76. Estos escenarios se basan en comandos y atributos que definen el comportamiento del escenario en cuestión. Dentro de algunos de estos comandos se definen las cabeceras SIP para determinar los mensajes a enviar
  El flujo de una sesión completa INVITE, según lo especificado en el RFC 3261, define a un endpoint iniciando la sesión con una petición INVITE (línea 11 a 33) hacia un

destinatario para el inicio de una sesión SIP. Es por ello, que se comienza la definición del escenario con el envío de un mensaje INVITE.

El comando <send> (línea 11) es utilizado para enviar al destinatario (especificado como DESTINATION entre los parámetros del bash script para el binario de SIPp, que es Kamailio) algún mensaje. Como se puede ver en la imagen, se hace uso de los parámetros [fieldX] que son tomados del archivo CSV, de resto se definen las distintas cabeceras con parámetros definidos por SIPp y los parámetros definidos por el archivo CSV para completar la cabecera SIP y se envíe correctamente.

Luego, se espera recibir un reply SIP con código 100 (línea 34) y luego 180 (línea 36) en caso de que sea exitoso. Debido a que se están usando mecanismos de autenticación por parte de Kamailio, se define que también se espera un reply SIP con código 407 y con parámetro auth="true" (línea 38), lo que especifica al escenario y a SIPp que espera que ese reply sea para iniciar un proceso de autenticación. Este reply se espera antes de recibir un código 100 o 180, es por ello que se coloca optional="true" debido a que es opcional recibir el código 100 o 180 (esto permite usar el mismo escenario en caso de no estar utilizando mecanismos de autenticación).

```
10: <scenario name="branch_client">
11:    <send retrans="500">
12:     <![CDATA[
13:       INVITE sip:[field3]@[field1]:[remote_port];transport=[transport] SIP/2.0
14:       Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
15:       From:
   <sip:[field0]@[field1]:[remote_port]>;transport=[transport];tag=[pid]SIPpTag00[f
   ield3]
16:       To: <sip:[field3]@[field1]:[remote_port]>
17:       Call-ID: [call_id]
18:       CSeq: 1 INVITE
19:       Contact: sip:[field0]@[local_ip]:[local_port]
20:       Max-Forwards: 70
21:       Subject: Performance Test
22:       Content-Type: application/sdp
23:       Content-Length: [len]
24:
25:       v=0
26:       o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
27:       s=-
28:       c=IN IP[media_ip_type] [media_ip]
29:       t=0 0
30:       m=audio [auto_media_port] RTP/AVP 8
31:       a=rtpmap:8 PCMA/8000
32:     ]]>
33:   </send>
34:   <recv response="100" optional="true" next="auth_done">
35:   </recv>
36:   <recv response="180" optional="true" next="auth_done">
37:   </recv>
38:   <recv response="407" auth="true">
```

Figura 9.76: Definición Inicial del Escenario SIPp

En caso de haber recibido el código 407, se define una cabecera SIP que envíe un mensaje ACK indicando que se entrará en el proceso de autenticación (Figura 9.77, línea 10 a 14), y luego se envía de vuelta un mensaje INVITE con CSeq 1 mayor al anterior para no confundirlo con una retransmisión. La diferencia entre el enviado anteriormente y el especificado acá es que, al saber que el código 407 recibido se trata de un proceso de autenticación (auth="true"), se define en la cabecera el parámetro [field2] (especificado como los parámetros de autenticación en el archivo CSV) para

```
10: <send>
11:     <![CDATA[
12:       ACK sip:[field3]@[field1]:[remote_port] SIP/2.0
13:       …
14:     ]]>
15:   </send>
16:   <send retrans="500">
17:     <![CDATA[
18:       INVITE sip:[field3]@[field1]:[remote_port];transport=[transport] SIP/2.0
19:       …
20:       Content-Type: application/sdp
21:       [field2]
22:       Content-Length: [len]
23:
24:       v=0
25:       o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
26:       s=-
27:       c=IN IP[media_ip_type] [media_ip]
28:       t=0 0
29:       m=audio [auto_media_port] RTP/AVP 8
30:       a=rtpmap:8 PCMA/8000
31:     ]]>
32:   </send>
```

formar la cabecera basándose en la cabecera de desafío en el reply de código 407 a fin de completar de manera exitosa el proceso de autenticación si la contraseña es correcta (Figura 9.77, línea 16 a 32).

Figura 9.77: Configuración del mecanismo de desafío MD5/Digest en SIPp

Una vez autenticado, se espera recibir replies con códigos 100 y 180 (Figura 9.78, línea 10 y 12) y luego el 200 OK que indica la finalización exitosa de inicio de sesión, éste es especificado con el parámetro rrs="true" (Figura 9.78, línea 14), esto es indicado debido a que se está utilizando Kamailio como un proxy con estado y se espera que el mensaje ACK de haber recibido el 200 OK sea recibido por Kamailio (mediante el mecanismo loose route). Esto implica que el mensaje ACK debe interpretar los mensajes Record-route de la cabecera 200 OK para mantener el enrutamiento de los mensajes SIP y enviarlo a Kamailio con la cabecera Route requerida (SIPp la específica al colocar en el escenario el parámetro [routes] como se puede ver en la Figura 9.78, línea 21).

```
10: <recv response="100" optional="true">
11: </recv>
12: <recv response="180" optional="true">
13: </recv>
14: <recv response="200" rrs="true" rtd="true">
15: </recv>
16:
17:   <send next="with_auth" crlf="true">
18:     <![CDATA[
19:       ACK [next_url] SIP/2.0
20:       Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
21:       [routes]
22:       …
23:     ]]>
24:   </send>
```

Figura 9.78: Configuración de Mecanismos Loose Route en Escenario SIPp

Ya iniciada la sesión comienza el envío de tráfico RTP, SIPp ofrece mecanismos para enviar, aunque de manera limitada, trafico RTP. La configuración del escenario puede verse en la Figura 9.79. Esto es realizado mediante archivos .pcap obtenidos mediante sniffing de trafico RTP con wireshark y transformado en un archivo .pcap. El archivo especificado es ofrecido por SIPp y consta de una conversación que se encuentra codificada con G711a (PCMA) que dura un total de 8 segundos. Este archivo .pcap es transmitido un total de 2 veces (línea 10 a 21) para simular una conversación de 16 segundos con tráfico RTP, y la culminación de la sesión con el envío de un mensaje BYE (línea 24), del cual se espera la recepción de un reply con código 200 OK para culminar el escenario (línea 30).

```
10: label id="with_auth"/>
11:    <nop>
12:     <action>
13:       <exec play_pcap_audio="pcap/g711a.pcap"/>
14:     </action>
15:   </nop>
16:   <pause milliseconds="8000"/>
17:   <nop>
18:     <action>
19:       <exec play_pcap_audio="pcap/g711a.pcap"/>
20:     </action>
21:   </nop>
22:   <pause milliseconds="8000"/>
23:   <pause milliseconds="1000"/>
24:   <send retrans="500">
25:     <![CDATA[
26:       BYE [next_url] SIP/2.0
27:       …
28:     ]]>
29:   </send>
30:   <recv response="200" rtd="true">
31:   </recv>
```

Figura 9.79: Configuración de Trafico RTP en el Escenario SIPp

Esta explicación describe la lógica de la mayoría de los escenarios formados para la realización de pruebas de estrés con SIPp a la implementación. Aunque la diferencia entre esta prueba y las otras es que se están utilizando dos endpoints con SIPp, por ello se define otro escenario con la misma estructura (bash script, archivo CSV y archivo XML) pero el escenario a definir se encarga de la recepción de peticiones SIP y

de enviar los reply requeridos La configuración inicial de dicho escenario puede verse en la Figura 9.80.

- **Archivo XML del endpoint receptor:** Sin entrar en el mismo detalle del escenario definido anteriormente, en este se puede ver que se espera por la recepción de un mensaje INVITE, una vez recibido se define el envío del reply con código 180 y 200 (línea 12 y 25). En este escenario se puede ver el uso del parámetro [last_*], esos parámetros son utilizados para que SIPp coloque la misma información que la recibida en la petición anterior, así se construye la cabecera SIP del reply como especificado en

```
10: recv request="INVITE" rrs="true" crlf="true">
11: </recv>
12:   <send>
13:     <![CDATA[
14:       SIP/2.0 180 Ringing
15:       [last_Via:]
16:       [last_From:]
17:       [last_To:];tag=[call_number]
18:       [last_Call-ID:]
19:       [last_CSeq:]
20:       [last_Record-route:]
21:       Contact: <sip:[field0]@[local_ip]:[local_port];transport=[transport]>
22:       Content-Length: 0
23:     ]]>
24:   </send>
25:   <send>
26:     <![CDATA[
27:       SIP/2.0 200 OK
28:       [last_Via:]
29:       [last_From:]
30:       [last_To:];tag=[call_number]
31:       [last_Call-ID:]
32:       [last_CSeq:]
33:       [last_Record-route:]
34:       Contact: <sip:[field0]@[local_ip]:[local_port];transport=[transport]>
35:       Content-Type: application/sdp
36:       Content-Length: [len]
37:       …
38:     ]]>
39:   </send>
```

el RFC 3261

Figura 9.80: Definición Inicial de Escenario SIPp actuando como UAS

Con este punto, se culmina la descripción de las actividades realizadas. En relación a lo explicado a lo largo de este documento, es importante considerar la definición de escenarios y realización de pruebas a la arquitectura que comprueben la fiabilidad y eficiencia del mismo. El próximo punto expresa aquellos escenarios de pruebas y comportamiento que se esperan por parte de la arquitectura, con la descripción de las pruebas a realizar a la misma.

10.Escenarios de Prueba y Comportamientos Esperados

Para la realización de las pruebas, tanto de mecanismos de contingencia como pruebas de estrés, se elaboraron los siguientes escenarios de pruebas, los cuales son divididos entre aquellos que serán utilizados para verificar el comportamiento de los mecanismos de contingencia, y los que serán utilizados para la realización de las pruebas de estrés requeridas. Además de eso, la descripción de los comportamientos esperados de los mismos y las pruebas a realizar, las que serán analizadas y descritas en el próximo capítulo.

10.1 Escenarios de Prueba

En este punto se describen los distintos escenarios de prueba a realizar para los mecanismos principales que ofrecen alta disponibilidad y escalabilidad a la arquitectura.

10.1.1 Escenarios de Mecanismos de Contingencia

Para verificar el comportamiento de los mecanismos de contingencia y los tiempos de respuesta del mismo se proponen los siguientes escenarios:

- **Escenario 1 – Caso de falla de un Componente PBX:** El siguiente escenario, descrito en la Figura 10.1, propone la situación de la caída de una instancia que se encuentra prestando el servicio de Asterisk, como se puede ver en la siguiente imagen:

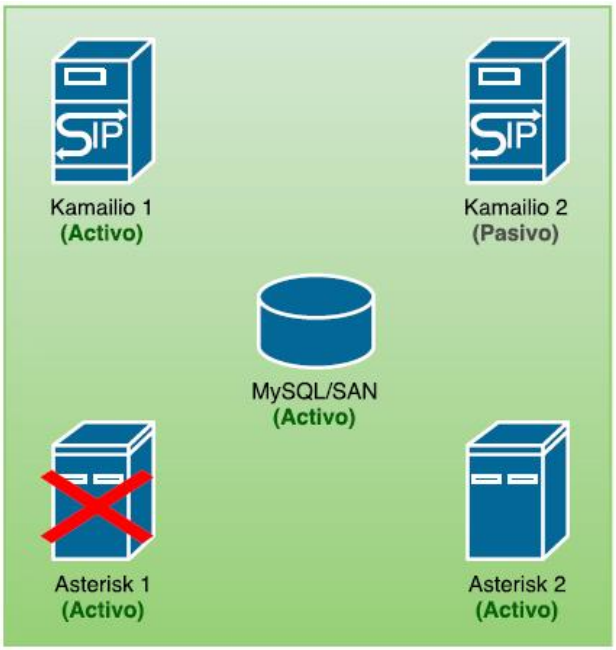


Figura 10.1: Diagrama del Escenario 1 para los Mecanismos de Contingencia

- **Escenario 2 – Caso de falla de un Componente Proxy/Registrar:** El siguiente escenario, descrito en la Figura 10.2, propone la situación de la caída de una

instancia que se encuentra prestando el servicio de Kamailio y RTPProxy, como se puede ver en la siguiente imagen:

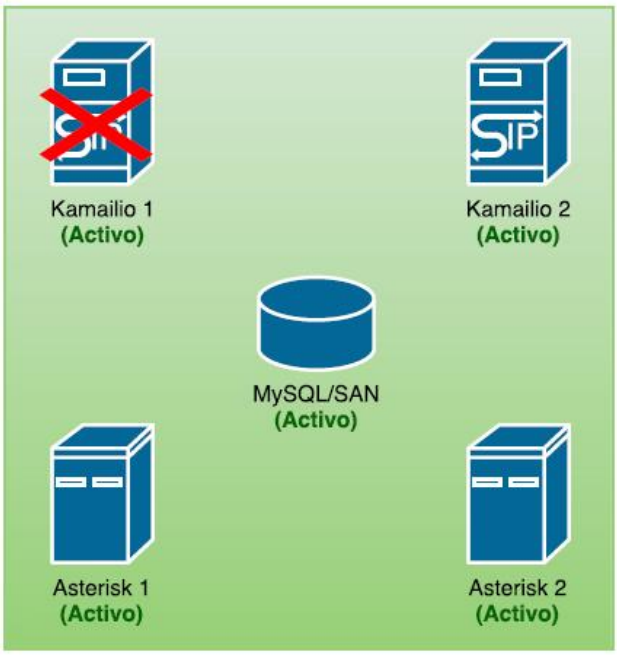


Figura 10.2: Diagrama del Escenario 2 para los Mecanismos de Contingencia

- **Escenario 3 – Caso de falla de un Componente Proxy/Registrar y uno PBX:** El siguiente escenario, descrito en la Figura 10.3, propone la situación de la caída de una instancia que se encuentra prestando el servicio de Kamailio y RTPProxy, y otro que se encuentra prestando el servicio de Asterisk, como se puede ver en la siguiente imagen:

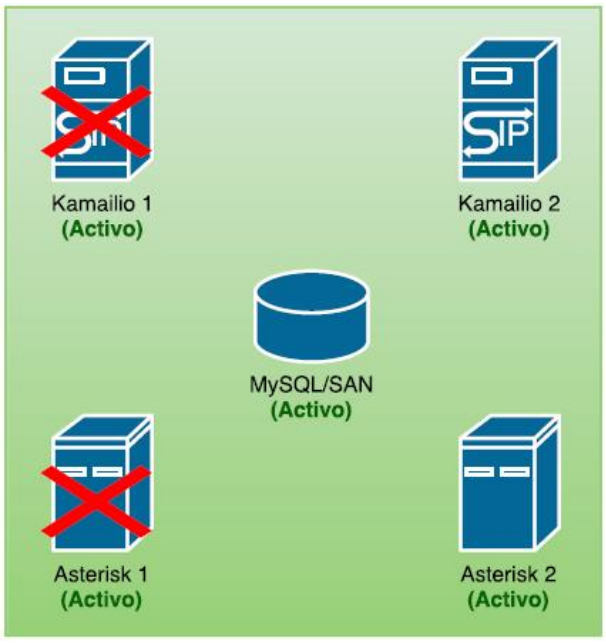


Figura 10.3: Diagrama del Escenario 3 para los Mecanismos de Contingencia

10.1.2 Escenarios de Pruebas de Estrés

Para verificar el comportamiento de la plataforma en escenarios de alta concurrencia de flujos de tráfico SIP (transacciones INVITE y REGISTER) y RTP se proponen los siguientes escenarios:

- **Escenario 1 – Todos los Componentes Activos:** El siguiente escenario, descrito en la Figura 10.4, propone la realización de pruebas de estrés en un escenario donde todos los componentes se encuentran en buen estado para verificar la escalabilidad de la arquitectura como tal. El escenario se muestra en la siguiente imagen:

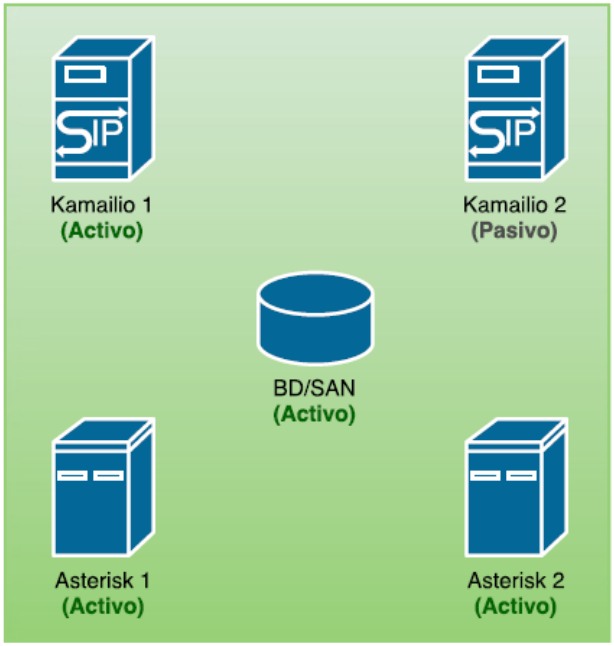


Figura 10.4: Diagrama del Escenario 1 para las Pruebas de Estrés

- **Escenario 2 – Con un Componente PBX Inactivo:** El siguiente escenario, descrito en la Figura 10.5, propone la realización de pruebas de estrés en un escenario donde uno de los componentes que se encuentran prestando el servicio de Asterisk se encuentre inactivo. El escenario se muestra en la siguiente imagen:

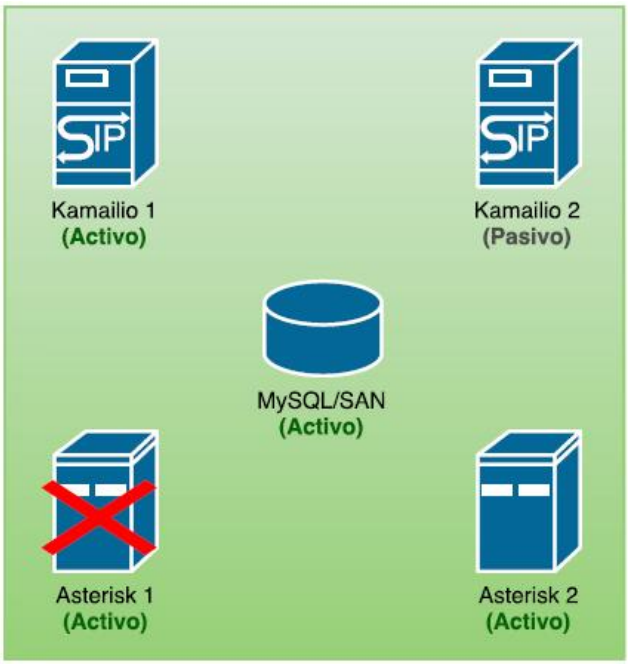


Figura 10.5: Diagrama del Escenario 2 para las Pruebas de Estrés

- **Escenario 3 – Todos los Componentes Activos más Solventar Problema de NAT Traversal:** El siguiente escenario, descrito en la Figura 10.6, propone la realización de pruebas de estrés en un escenario donde todos los componentes

se encuentran en buen estado para verificar la escalabilidad de la arquitectura en escenarios que requieran solventar el fenómeno de NAT Traversal. El escenario se muestra en la siguiente imagen:

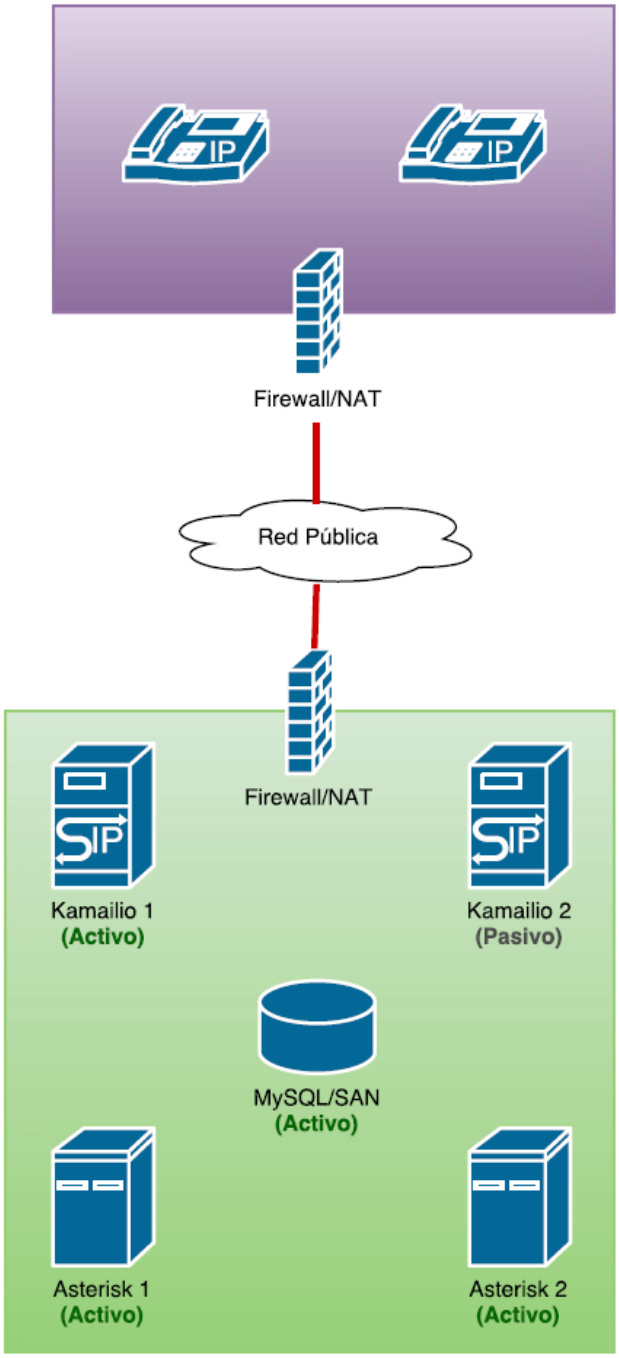


Figura 10.6: Diagrama del Escenario 3 para las Pruebas de Estrés

10.2 Comportamientos Esperados y Pruebas a Realizar

En esta sección se hace mención a las distintas pruebas que se van a realizar para cada tipo de escenario a tocar y los comportamientos esperados de algunos de ellos.

10.2.1 Mecanismos de Contingencia

En este trabajo se manejan dos tipos de mecanismos, uno de ellos basado en las funciones de balanceo de carga de Kamailio cuando ocurren fallas a nivel de los nodos de Asterisk que forman parte de la arquitectura. Y el otro se basa en las posibles fallas a nivel de los servicios y nodos, los cuales son gestionados por Pacemaker/Corosync. El comportamiento de dichos mecanismos varía, principalmente, dependiendo del estado de una transacción, estas pruebas se definen a continuación:

- **Prueba 1:** Envío de transacción INVITE justo luego de ocurrir una falla para la aplicación de mecanismos de contingencia
- **Prueba 2:** Envío de transacción INVITE antes de ocurrir una falla para la aplicación de mecanismos de contingencia sin todavía haberse finalizado el inicio de sesión INVITE
- **Prueba 3:** Envío de transacción INVITE antes de ocurrir una falla para la aplicación de mecanismos de contingencia pero ya iniciado la sesión INVITE (establecida sesión de mensajes multimedia)

Es importante destacar que los mecanismos de contingencia automatizados aplicables, principalmente por Kamailio mediante los mecanismos de failover transaccionales, son basados en transacciones SIP activas. Esto significa que, para la Prueba 3, en caso de ocurrir fallas en alguno de los componentes de la arquitectura (Proxy/Registrar o PBX), por donde el tráfico RTP de dicha sesión forme parte (en caso de aplicar), se requiere de la interacción del usuario para establecer de vuelta la llamada. Por ello, dicha prueba no posee tanta relevancia en los resultados de las pruebas a analizar.

10.2.1.1 Mecanismos de Failover Transaccionales

Este mecanismo es tratado por la capacidad de Kamailio de mantener estado de las transacciones SIP y tomar decisiones en base a la falta o error de respuesta por parte de los nodos de Asterisk que forman parte de la arquitectura. Para este tipo de mecanismo, existen dos situaciones en las cuales se aplica failover para peticiones de inicio de sesiones SIP con INVITE. Estos mecanismos de failover ocurren cuando hay presencia de un error en alguno de estos nodos de Asterisk por un breve período de tiempo hasta que, mediante mecanismos de health checks, Kamailio perciba la falla del nodo en cuestión, o cuando la primera ejecución de estos mecanismos ocurra, así marcando así al nodo como inactivo.

Situación 1

Este primer escenario ocurre cuando una petición INVITE intenta ser transmitida a un nodo de Asterisk que recién falló y hay una falta de respuesta del mismo, como fue explicado en la Prueba 1. En el punto 9.4.5.5, se hace mención a la colocación de un timer que especifica el tiempo (4 segundos) de espera maximo antes de recibir alguna respuesta para esta transacción INVITE. En caso de no responder, Kamailio retransmitirá la petición INVITE al otro nodo de Asterisk que se encuentra activo y marcará al fallido como inactivo. La Figura 10.7 muestra el flujo comentado en este párrafo:

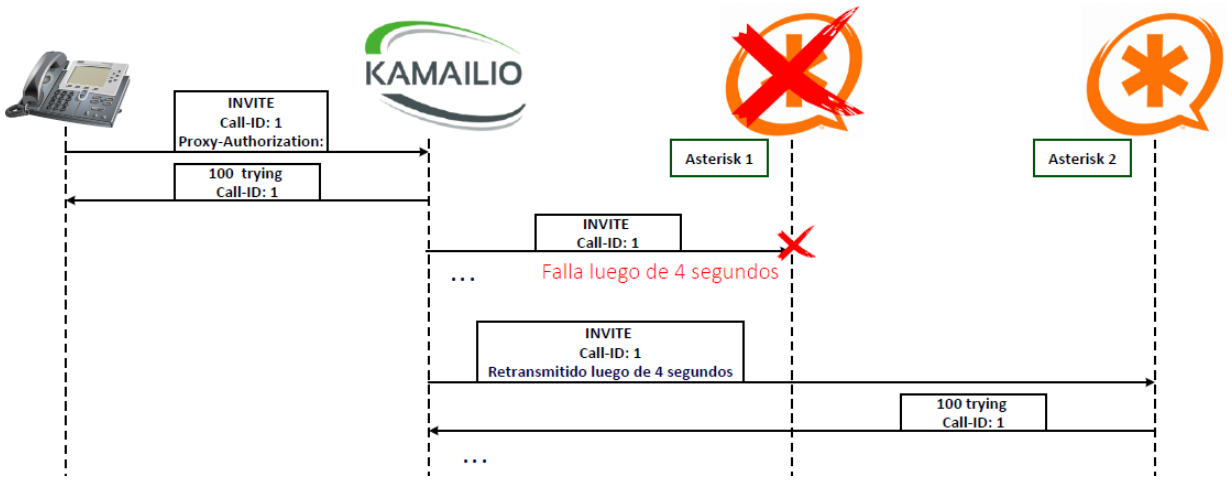


Figura 10.7: Mecanismo de Failover en caso de no Recibir Respuesta a la Petición INVITE

Situación 2

El otro escenario se basa en la misma idea debido a que los mecanismos de failover que aplica el módulo de balanceo de carga de Kamailio son a nivel de peticiones transaccionales, pero en este caso se aplica un poco más de detalle. Este es aplicable en la Prueba 2.

El RFC 3261 define, por defecto, un timer de 180 segundos como tiempo máximo de espera por algún componente SIP intermedio para la recepción de una respuesta final (2xx, 4xx, 5xx, etc) de una transacción INVITE abierta, pero, aunque no especificado en dicho RFC, algunos UACs definen un timer igual para dicha transacción de un total de 120 segundos o más. Esto significa que si alguno de los componentes inmersos en la arquitectura manejan este timer por defecto, a los 180 segundos de no haber recibido la respuesta final de la transacción INVITE, Kamailio intentaría aplicar el mecanismo de failover hacia otro nodo de Asterisk pero el endpoint cerraría dicha transacción. Todo esto conlleva a disminuir el timer de espera de respuesta final tanto en Kamailio como en Asterisk, donde, en Asterisk debe ser igual o menor al especificado en Kamailio.

Ahora, este escenario de mecanismo de failover se toma en cuenta luego de la recepción del mensaje 100 Trying de la primera sesión (en el escenario está marcado como sesión con Call-ID igual a 1). Si en el proceso especificado en la próxima imagen (puede ser incluso antes), Kamailio no recibe ninguna respuesta luego de los 60 segundos (usualmente se debe a caída del servidor con Asterisk), Kamailio retransmitirá dicha transacción INVITE al otro nodo de Asterisk que se encuentra activo y marcará al fallido como inactivo. La Figura 10.8 muestra el escenario planteado:

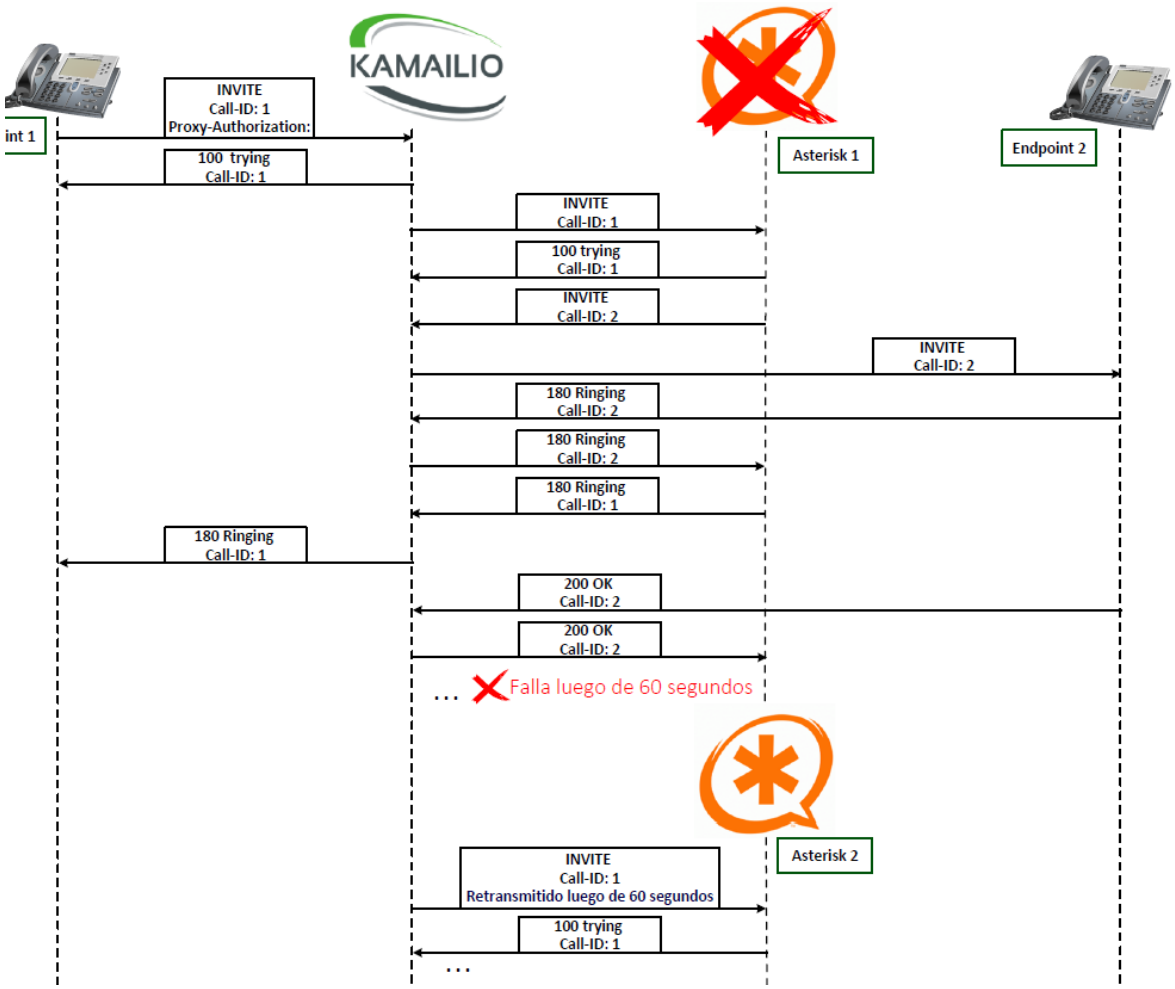


Figura 10.8: Mecanismo de Failover en caso de no Recibir Respuesta Final a la Petición INVITE

Las pruebas de demostración de mecanismos de failover transaccionales esperan que los resultados sean como los especificados en la Tabla 10.1:

Escenario	Comportamiento	Tiempos de Repuesta
Escenario 1	• Interacción de Usuario: Ninguna • Contingencia: Retransmisión de inicio de sesión INVITE luego de cumplir los tiempos de respuesta	• Situación 1: 4 segundos • Situación 2: 60 segundos

Tabla 10.1: Resultados Esperados de los Mecanismos de Failover Transaccionales

10.2.1.2 Mecanismos de Failover a Nivel de Servicios y Nodos

Este mecanismo de failover hace mención al definido mediante los esquemas de clustering utilizando las herramientas Pacemaker/Corosync, el cual es utilizado en el esquema de clustering de componentes Proxy/Registrar y de componentes PBX.

Esquemas de Clustering de Componentes Proxy/Registrar

Como fue especificado en el punto 9.4.5.4, dicho esquema consta de 6 recursos Activos/Pasivos y 2 dispositivos de fencing que se encuentran clonados y activos en ambos nodos (debido al delay que generan al iniciar). Este esquema es formado por dos nodos, kam1 y kam2, en el ejemplo de la Figura 10.9 se puede notar que el nodo kam2 se encuentra actuando como nodo activo y kam1 como pasivo.

```
[root@kam2 ~]# pcs resource
 ClusterIP      (ocf::heartbeat:IPaddr2):       Started kam2vs3
 ClusterIP2     (ocf::heartbeat:IPaddr2):       Started kam2vs3
 kamailioetcfs  (ocf::heartbeat:Filesystem):    Started kam2vs3
 Clone Set: fence_kam2_xvm-clone [fence_kam2_xvm]
     Started: [ kam1vs3 kam2vs3 ]
 Master/Slave Set: kamailioetcclone [kamailioetc]
     Masters: [ kam2vs3 ]
     Slaves: [ kam1vs3 ]
 Clone Set: fence_kam1_xvm-clone [fence_kam1_xvm]
     Started: [ kam1vs3 kam2vs3 ]
 kamailiocluster        (ocf::heartbeat:kamailio):      Started kam2vs3
 rtpproxycluster        (ocf::heartbeat:rtpproxy):      Started kam2vs3
```

Figura 10.9: Estado del Cluster Proxy/Registrar antes de la aplicación de Failover

Debido a ello, kam2 es el que se encuentra proveyendo los servicios mientras kam1 está a la expectativa de alguna posible falla en kam2 para la migración de los servicios. Esta migración es transparente y no debe afectar a la información de, por ejemplo, identificación de los nodos de Asterisk que forman parte de la arquitectura, la localización de los usuarios SIP anteriormente registrados en la arquitectura, etc.

En caso de ocurrir una falla en kam2, o en algún recurso gestionado por kam2, los servicios serán migrados a kam1 para proveerlos él. Para la demostración del mecanismo de failover, se enviará una señal desde el nodo kam1 para aplicar fencing en el nodo kam2 simulando así un error a nivel del esquema de clustering como se ve en la Figura 10.10.

```
[root@kam1 ~]# pcs stonith fence kam2vs3
Node: kam2vs3 fenced
```

Figura 10.10: Resultado de la Aplicación de Fencing al Nodo kam2

Una vez aplicado el fencing y culminado exitosamente en el nodo kam2, los servicios comenzarán a iniciarse en el nodo kam1 como lo muestra la Figura 10.11.

```
[root@kam1 ~]# pcs resource
 ClusterIP      (ocf::heartbeat:IPaddr2):       Started kam1vs3
 ClusterIP2     (ocf::heartbeat:IPaddr2):       Started kam1vs3
 kamailioetcfs  (ocf::heartbeat:Filesystem):    Started kam1vs3
 Clone Set: fence_kam2_xvm-clone [fence_kam2_xvm]
     Started: [ kam1vs3 ]
     Stopped: [ kam2vs3 ]
 Master/Slave Set: kamailioetcclone [kamailioetc]
     Masters: [ kam1vs3 ]
     Stopped: [ kam2vs3 ]
 Clone Set: fence_kam1_xvm-clone [fence_kam1_xvm]
     Started: [ kam1vs3 ]
     Stopped: [ kam2vs3 ]
 kamailiocluster        (ocf::heartbeat:kamailio):      Started kam1vs3
 rtpproxycluster        (ocf::heartbeat:rtpproxy):      Started kam1vs3
```

Figura 10.11: Estado del Cluster Proxy/Registrar luego de la aplicación de Failover

Las pruebas de demostración de mecanismos de failover del esquema de clustering Proxy/Registrar esperan que los resultados sean como los especificados en la Tabla 3.2:

Escenario	Comportamiento	Tiempos de Repuesta

Escenario 2	• Interacción de Usuario: Replicar la petición inicial en caso de estar en proceso de establecimiento de la sesión • Contingencia: Restablecimiento del Servicio	• Aplicación de Fencing: Pocos segundos • Errores de Servicios o Nodos: Pocos segundos
Escenario 3	• Interacción de Usuario: Replicar la petición inicial en caso de estar en proceso de establecimiento de la sesión • Contingencia: Restablecimiento del Servicio	• Aplicación de Fencing: Pocos segundos • Errores de Servicios o Nodos: Pocos segundos

Tabla 10.2: Resultados Esperados caso Mecanismos de Contingencia de Proxy/Registrar

Esquemas de Clustering de Componentes PBX

Como fue especificado en el punto 9.4.4.3, dicho esquema consta de 4 recursos clonados y 2 dispositivos de fencing también clonados. Este esquema es formado por dos nodos, pbx1 y pbx2 y su finalidad radica en la prevención de corrupción de datos que se encuentran en el almacenamiento compartido y en la capacidad de monitorear el estado del servicio de Asterisk con mayor granularidad.

```
[root@pbx1 ~]# pcs resource
 Clone Set: dlm-clone [dlm]
     Started: [ pbx1vs3 pbx2vs3 ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ pbx1vs3 pbx2vs3 ]
 Clone Set: asteriskfs-clone [asteriskfs]
     Started: [ pbx1vs3 pbx2vs3 ]
 Clone Set: asterisk-clone [asterisk]
     Started: [ pbx1vs3 pbx2vs3 ]
 Clone Set: fence_pbx1_xvm-clone [fence_pbx1_xvm]
     Started: [ pbx1vs3 pbx2vs3 ]
 Clone Set: fence_pbx2_xvm-clone [fence_pbx2_xvm]
     Started: [ pbx1vs3 pbx2vs3 ]
```

Figura 10.12: Estado del Cluster PBX antes de la aplicación de la Contingencia

A diferencia del esquema de cluster de Kamailio, aquí no se recurre a la migración de servicios en caso de fallas. Simplemente, dichos servicios se pararán dependiendo de las restricciones de orden y colocación que posean. Así, el nodo no se encontrará proveyendo el servicio de Asterisk (salvo, que sea en base al primer error de monitoreo en el mencionado recurso, que implica el reinicio de dicho servicio). El resultado puede verse en la Figura 10.14. Para la demostración del mecanismo, se enviará una señal desde el nodo pbx1 para aplicar fencing en el nodo pbx2 simulando así un error a nivel del esquema de clustering. Esto puede verse en la Figura 10.13.

```
[root@pbx1 ~]# pcs stonith fence pbx2vs3
Node: pbx2vs3 fenced
```

Figura 10.13: Resultado de la Aplicación de Fencing al Nodo pbx2

```
[root@pbx1 ~]# pcs resource
 Clone Set: dlm-clone [dlm]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
 Clone Set: asteriskfs-clone [asteriskfs]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
 Clone Set: asterisk-clone [asterisk]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
 Clone Set: fence_pbx1_xvm-clone [fence_pbx1_xvm]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
 Clone Set: fence_pbx2_xvm-clone [fence_pbx2_xvm]
     Started: [ pbx1vs3 ]
     Stopped: [ pbx2vs3 ]
```

Figura 10.14: Estado del Cluster PBX luego de la aplicación de la Contingencia

Las pruebas de demostración de mecanismos de failover del esquema de clustering PBX esperan que los resultados sean como los especificados en la Tabla 10.2:

Escenario	Comportamiento	Tiempos de Repuesta
Escenario 1	• Interacción de Usuario: Replicar la petición inicial en caso de estar en proceso de establecimiento de la sesión • Contingencia 1: En caso de primera falla de monitoreo del recurso de Asterisk en un nodo, Reinicio del Servicio • Contingencia 2: En cualquier otro caso, Parar el Servicio en el nodo donde falla	• Aplicación de Fencing: Pocos segundos • Errores de Servicios o Nodos: Pocos segundos
Escenario 3	• Interacción de Usuario: Replicar la petición inicial en caso de estar en proceso de establecimiento de la sesión • Contingencia 1: En caso de primera falla de monitoreo del recurso de Asterisk en un nodo, Reinicio del Servicio • Contingencia 2: En cualquier otro caso, Parar el Servicio en el nodo donde falla	• Aplicación de Fencing: Pocos segundos • Errores de Servicios o Nodos: Pocos segundos

Tabla 10.3: Resultados Esperados caso Mecanismos de Contingencia PBX

Para los mecanismos de contingencia, las pruebas a realizar serán hechas también con SIPp estableciendo sesiones INVITE simulando ambos endpoints. A diferencia de aquellas pruebas que serán realizadas para las pruebas de estrés, éste se encargará del establecimiento de una llamada solamente, pero monitoreando los tiempos de respuesta entre algunos mensajes SIP provenientes del establecimiento de sesión INVITE que forman parte crucial para la demostración de los mecanismos de alta disponibilidad explicados a lo largo de la Sección 10.2.1. Estos tiempos de respuestas son monitoreados de parte de SIPp con la utilización del parámetro Response Timer Duration (RTD). La Figura 10.15 demuestra el flujo de mensajes de los escenarios configurados para cada endpoint y los RTDs utilizados para la medición de la respuesta de dichos mensajes.

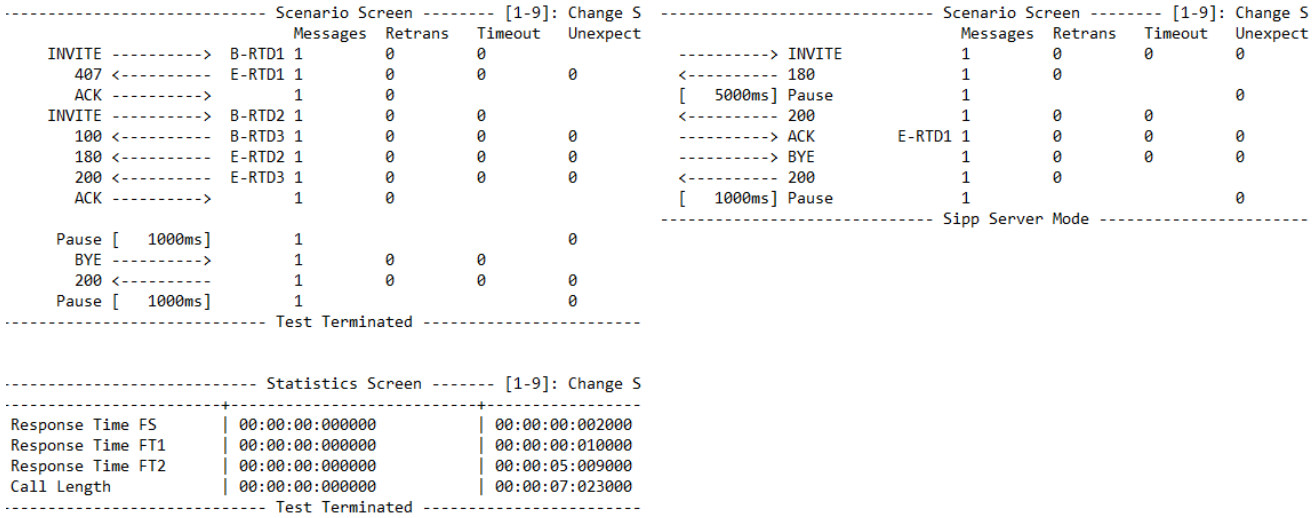


Figura 10.15: Escenarios en SIPp para las Pruebas de Mecanismos de Contingencia

- **Response Time FS:** Este es el timer entre mensajes SIP que permite medir, a percepción del usuario, cuanto tiempo dura la aplicación del failover a nivel de servicio de los componentes Proxy/Registrar
- **Response Time FT1:** Este es el timer entre mensajes SIP que permite medir, a percepción del usuario, cuanto tiempo dura la aplicación del failover a nivel transaccional de la situación 1 explicada en la Sección 10.2.1.1
- **Response Time FT2:** Este es el timer entre mensajes SIP que permite medir, a percepción del usuario, cuanto tiempo dura la aplicación del failover a nivel transaccional de la situación 2 explicada en la Sección 10.2.1.1
- La demostración de failover a nivel de servicios del componente PBX no es perceptible por el usuario debido a que los mecanismos de failover transaccionales se encargan de ello en caso de aplicar mecanismos de contingencia

10.3 Pruebas de Estrés

En este trabajo se plantean X pruebas que pondrán a la arquitectura en ambientes de alta concurrencia para comprobar la existencia de alguna mejoría, contando con las limitaciones de recursos que dispone la implementación, dentro de la arquitectura. Dichas pruebas son realizadas con SIPp y son descritas a continuación.

10.3.1 Prueba de Flujo de Mensajes INVITE entre dos Endpoints

Esta prueba consiste en la realización de llamadas, como fue explicado en el punto 9.3.5.1, pero con la diferencia que los Endpoint definidos en la Figura 9.6 son escenarios con SIPp. Las construcciones de estos escenarios fueron explicadas en el punto 9.4.6.1 y permiten la alta concurrencia de llamadas adecuadas para distintos escenarios. Las pruebas a realizar para medir la arquitectura son definidos en la Tabla 10.4:

Escenarios	Pruebas	Tasa de Llamadas	Límite de Llamadas Activas	Tiempo de Duración de las Llamadas	Tiempo de Duración de la Prueba

Escenario 1	**Prueba 1**	300 cps	2000 Llamadas	6 segundos	1 hora
	Prueba 2	300 cps	2000 Llamadas	7 segundos	1 hora
	Prueba 3	350 cps	2000 Llamadas	5 segundos	1 hora
	Prueba 4	350 cps	2000 Llamadas	6 segundos	1 hora
Escenario 2	**Prueba 1**	175 cps	1000 Llamadas	6 segundos	1 hora
	Prueba 2	200 cps	1000 Llamadas	5 segundos	1 hora
	Prueba 3	200 cps	1200 Llamadas	5 segundos	1 hora
	Prueba 4	200 cps	1200 Llamadas	6 segundos	1 hora
Escenario 3	**Prueba 1**	300 cps	2000 Llamadas	6 Segundos	1 hora
	Prueba 2	250 cps	1600 Llamadas	6 Segundos	1 hora
	Prueba 3	250 cps	2000 Llamadas	7 Segundos	1 hora
	Prueba 4	275 cps	2000 Llamadas	7 Segundos	1 hora

Tabla 10.4: Tasa de Llamadas de la Prueba de Flujo de Mensajes INVITE entre dos Endpoints

10.3.2 Prueba de Flujo de Mensajes REGISTER

Esta prueba consiste en la realización de registros de usuarios SIP hacia la arquitectura de telefonía. Esto permite medir a la instancia que actúa como Registrar Server (en este caso Kamailio). Las pruebas a realizar para medir la arquitectura son definidos en la Tabla 10.5 y el flujo de mensajes definidos por el escenario SIPp se puede ver en la Figura 10.16:

```
                                Messages  Retrans   Timeout   Unexpected-Msg
REGISTER ---------->            1         0         0
     200 <----------            0         0         0         0
     401 <----------            1         0         0         0
REGISTER ---------->            1         0         0
     200 <----------            1         0         0         0
```

Figura 10.16: Prueba de Flujo de Mensajes REGISTER

Las pruebas a realizar para medir al Registrar Server son definidos en la Tabla 10.5:

Escenarios	Pruebas	Tasa de Transacciones	Cantidad de Usuarios SIP	Tiempo de Duración
Escenario 1	**Prueba 1**	500 tps	10000 usuarios	1 hora
	Prueba 2	750 tps	10000 usuarios	1 hora
	Prueba 3	1000 tps	10000 usuarios	1 hora
Escenario 3	**Prueba 1**	500 tps	10000 usuarios	1 hora
	Prueba 2	750 tps	10000 usuarios	1 hora
	Prueba 3	1000 tps	10000 usuarios	1 hora

Tabla 10.5: Tasa de Llamadas de las Pruebas de Estrés del Flujo de Mensajes REGISTER

En este capítulo se definieron las pruebas a realizar, los comportamientos esperados de las distintas pruebas y los distintos escenarios donde las pruebas son efectuadas. Estas pruebas conllevaron a algunos resultados, de los cuales es necesario analizar dichos resultados, basándose en el comportamiento esperado y los distintos escenarios donde fueron realizados dichas pruebas. Esto es abordado en el próximo capítulo.

11.Verificación y Análisis de Resultados

Este capítulo hace mención a la verificación y análisis de los resultados obtenidos en los escenarios de prueba descritos en el capítulo 10 de este trabajo. La misma se divide en dos tipos de pruebas distintos, pruebas de mecanismos de contingencia y pruebas de estrés.

Las pruebas son realizadas desde una maquina corriendo escenarios creados con SIPp hacia la arquitectura, como lo muestran ambos diagramas descritos en la Figura 11.1.

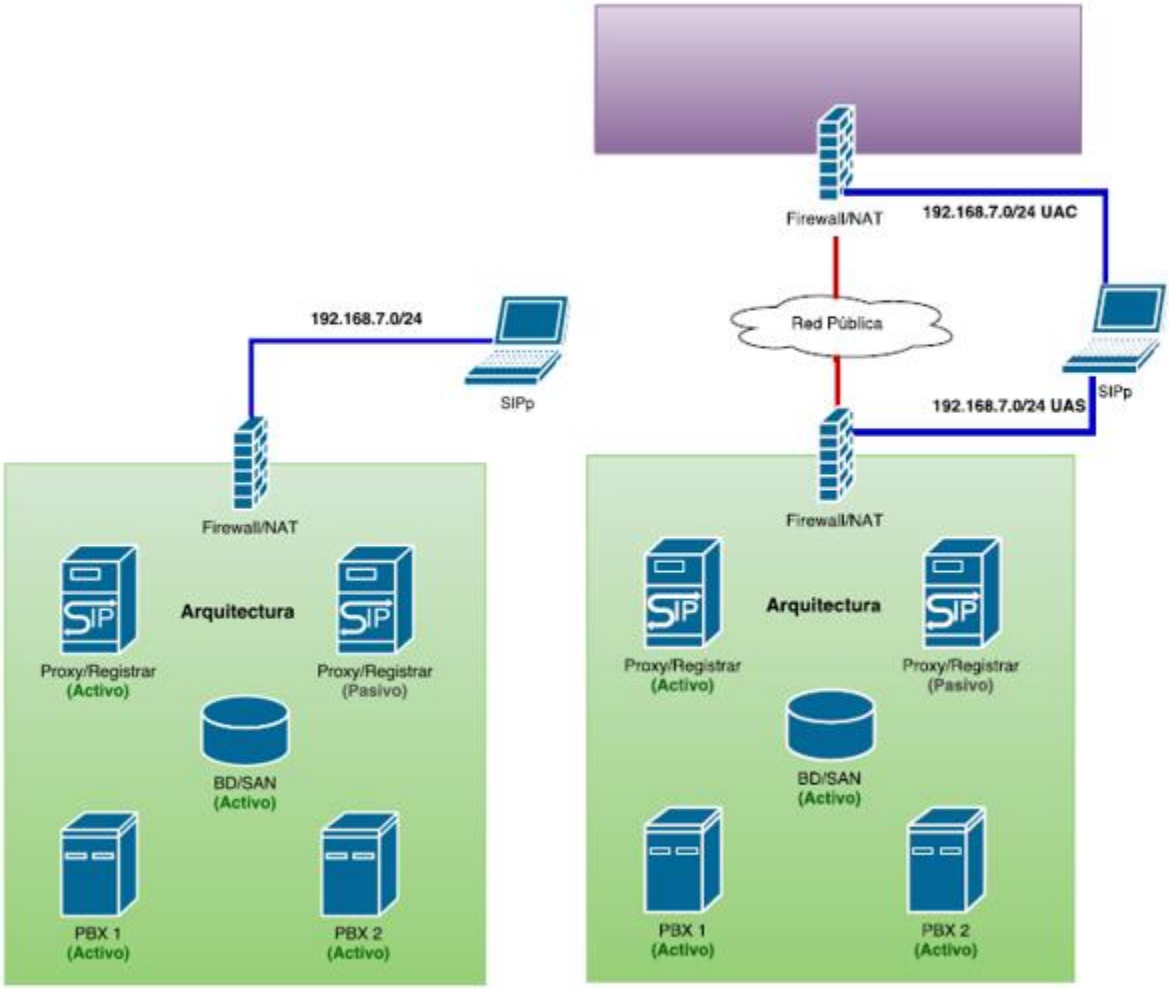


Figura 11.1: Diagramas Descriptivos de los Entes que forman parte de las Pruebas

11.1 Pruebas de Mecanismos de Contingencia

En estas pruebas se obtuvieron resultados interesantes, en su mayoría cumplieron con los comportamientos esperados, y otros inesperados pero positivos. Para este caso, las pruebas son divididas dependiendo de los tipos de mecanismos de failover, los cuales son:

11.1.1 Mecanismos de Failover Transaccionales

Los resultados de las pruebas, basándose en que dichas pruebas son realizadas desde la perspectiva del usuario, obtuvieron un comportamiento esperado. Los mecanismos descritos en esta Sección solamente son aplicables en el Escenario de prueba 1 y, dependiendo de si el servicio de Kamailio es migrado correctamente antes de la culminación del servicio de Asterisk en el nodo caído, es aplicable en el Escenario 3. De la misma manera, son sólo aplicables en la Prueba 1 y Prueba 2.

Escenarios	Prueba	Situación	Interacción de Usuario	Contingencia	Tiempos de Repuesta
Escenario 1	**Prueba 1**	**Situación 1**	Ninguna	Retransmisión de inicio de sesión INVITE luego de cumplir los tiempos de respuesta	00:00:03:974000 (3.974 segundos)
	Prueba 2	**Situación 2**	Ninguna	Retransmisión de inicio de sesión INVITE luego de cumplir los tiempos de respuesta	00:01:05:049000 (65.049 segundos)

Tabla 11.1: Resultados de las Pruebas de Mecanismos de Failover Transaccionales

Como se puede ver en la Tabla 11.1, los tiempos de respuesta a los mecanismos de failover transaccionales coinciden con los parámetros configurados en la aplicación de failover transaccional por Kamailio como se comentó en la Sección 10.2.1. En los resultados de la Prueba 2 se pueden observar 5.049 segundos extras de tiempo de respuesta con respecto a los configurados en Kamailio. Esto se debe a que se configura, en los escenarios de SIPp, una pausa de 5 segundos entre el envío de la respuesta 180 y la 200 como lo muestra la Figura 10.15. La razón estriba en que es un espacio de tiempo que habilita la posibilidad de aplicar las pruebas deseadas y además es considerado un comportamiento natural en cualquier comunicación entre dos usuarios SIP que formen parte de la arquitectura.

11.1.2 Mecanismos de Failover a Nivel de Servicios y Nodos

El resultado de las pruebas, basándose en que dichas pruebas son realizadas desde la perspectiva del usuario, obtuvo un comportamiento esperado. A diferencia de las pruebas realizadas para la Sección 11.1.1, las Pruebas 2 y 3 requieren de la interacción del usuario para establecer de vuelta una llamada.

Esquemas de Clustering de Componentes Proxy/Registrar

El resultado de las pruebas de mecanismos de failover en esquemas de clustering de componentes Proxy/Registrar se puede ver en la Tabla 11.2.

Escenario	Prueba	Interacción de Usuario	Contingencia	Fencing	Tiempos de Repuesta
Escenario 2	**Prueba 1**	Ninguna	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - 00:00:03:508000 Servicio - 00:00:03:495000
				Si	Usuario - 00:00:07:513000 Servicio - 00:00:07:387000
	Prueba 2	Ninguna	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - 00:00:03:519000 Servicio - 00:00:03:447000
				Si	Usuario - 00:00:07:518000 Servicio - 00:00:07:412000
	Prueba 3	Replicar establecimiento de llamada	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - Puede o no afectar el servicio Servicio - 00:00:03:482000
				Si	Usuario - Puede o no afectar el servicio Servicio - 00:00:07:398000
Escenario 3	**Prueba 1**	Ninguna	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - 00:00:07:470000 Servicio - 00:00:03:432000
				Si	Usuario - 00:00:11:499000 Servicio - 00:00:07:340000
	Prueba 2	Replicar establecimiento de llamada	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - Requiere Interacción Servicio - 00:00:03:388000
				Si	Usuario - Requiere Interacción Servicio - 00:00:07:455000
	Prueba 3	Replicar establecimiento de llamada	Servicios son migrados de un nodo que forma parte del esquema de clustering al otro.	No	Usuario - Requiere Interacción Servicio - 00:00:03:449000
				Si	Usuario - Requiere Interacción Servicio - 00:00:07:398000

Tabla 11.2: Resultados de las Pruebas de Mecanismos de Failover en Esquemas de Clustering de Componentes Proxy/Registrar

La mayoría de los resultados experimentaron comportamientos y tiempos de respuesta esperados. Sin embargo, hubo ciertos resultados inesperados sin ser necesariamente negativos. En los siguientes puntos se describen con mayor precisión el comportamiento de algunas de las pruebas que con la Tabla 11.2 no se logran describir a detalle.

- **Escenario 2 – Prueba 2:** En este caso, al ser enviado una respuesta 200 OK por parte de uno de los endpoints en plena migración, dicho mensaje es retransmitido por el

endpoint un total de 3 veces (4 en caso de fencing) hasta que el servicio de Kamailio es iniciado por el otro nodo para completar el establecimiento de la sesión SIP.

- **Escenario 2 – Prueba 3:** Para este caso, la comunicación de la sesión establecida presentará fallas en caso de que dicha comunicación incluya al fenómeno de NAT Traversal. De no ser este el caso, no presentará fallas en el servicio. El caso borde es que una transacción BYE por parte de los endpoints sea enviada en pleno proceso de migración, esto cortaría la comunicación sin que el otro endpoint y el nodo PBX reciban el mensaje BYE.
- **Escenario 3 – Prueba 1:** El tiempo de respuesta es de más de 7 segundos debido a que, al inicio de Kamailio, el estado de ambos servidores multimedia se encuentran en un estado llamado AX, el cual no es un estado definitivo de inactividad o actividad de dichos servidores multimedia, lo que implica en la aplicación de failover transaccional en la Situación 1 como fue explicado en la Sección 10.2.1. Los 7 segundos se debe a la suma entre el tiempo de la migración y la aplicación de failover transaccional, de igual manera como en el caso con fencing que ocurre luego de 11 segundos.
- **Escenario 3 – Prueba 3:** En estos casos, el tráfico RTP de la comunicación establecida se corta y, en caso de que la transacción BYE sea enviada por uno de los endpoint, el otro no lo recibirá.

Esquemas de Clustering de Componentes PBX

Como fue mencionado en la Sección 10.2.1.2, este esquema es utilizado para prevenir la corrupción de datos en el almacenamiento compartido y la capacidad de monitorear el estado del servicio de Asterisk con mayor granularidad. Los mecanismos de failover, para la perspectiva del usuario, son gestionados mediante los mecanismos de failover transaccionales en caso de fallas en los componentes PBX. Por lo tanto, las pruebas con los escenarios de SIPp construidos no permiten evaluar los tiempos de respuesta en caso de fallas en el servicio de algunos de estos componentes. El resultado de las pruebas de mecanismos de failover en esquemas de clustering de componentes PBX se puede ver en la Tabla 11.3.

Escenario	Contingencia	Fencing	Tiempos de Repuesta
Escenario 1	• Contingencia 1: En caso de primera falla de monitoreo del recurso de Asterisk en un nodo, Reinicio del Servicio • Contingencia 2: En cualquier otro caso, Parar el Servicio en el nodo donde falla	No	00:00:01:215000
		Si	00:00:03:552000
Escenario 3	• Contingencia 1: En caso de primera falla de monitoreo del recurso de Asterisk en un nodo, Reinicio del Servicio • Contingencia 2: En cualquier otro caso, Parar el Servicio en el nodo donde falla	No	00:00:01:208000
		Si	00:00:03:618000

Tabla 11.3: Resultados de las Pruebas de Mecanismos de Failover en Esquemas de Clustering de Componentes PBX<

Los resultados de las pruebas de mecanismos de contingencia fueron positivos. Sin embargo, es relevante destacar que dichos tiempos de respuesta fueron medidos justo después de que la arquitectura perciba una falla y aplique el mecanismo. Esto significa, que dichos tiempos de respuesta se encuentran en el peor caso una vez presenciado alguna falla que indique aplicar dichos mecanismos, y en un caso medio, si ocurre alguna falla en el servicio y el esquema no ha monitoreado el estado de falla de dicho servicio. Muchos de los recursos monitoreados por los distintos esquemas fueron configurados con intervalos de monitoreo de alrededor de 10 segundos, dichos intervalos pueden ser considerablemente más bajos debido a que sus mecanismos de monitoreo son más rápidos y sencillos, pero a su vez son servicios que en esta arquitectura no han sufrido de fallas concurrentes que eviten un buen funcionamiento del mismo.

Por otro lado, están los recursos de Kamailio y Asterisk que son aquellos que principalmente llevan el mayor peso dentro de la arquitectura. Por ello, es necesario un monitoreo más granular y precisado, con intervalos bien estudiados para la percepción de las fallas en dichos servicios lo antes posibles pero, a su vez, evitar una percepción errónea de una falla estando el recurso en buen estado. Dichos inconvenientes se han logrado observar a lo largo de ciertas pruebas y, como se verá en la Sección 11.2, se realizan algunas modificaciones para evitar dichos inconvenientes.

11.2 Pruebas de Estrés

A diferencia de las pruebas realizadas en la Sección 11.1, donde los resultados fueron positivos, el resultado de las pruebas explicadas en esta sección son altamente dependientes de la cantidad de recursos de hardware y el desarrollo de la topología escogida. Muchas de las razones por las cuales se tomaron algunas de estas decisiones se basan en: la imposibilidad de desarrollar distintas alternativas existentes en el ambiente donde es desplegado y en la misma esperanza de poder obtener mejores resultados.

Como fue mostrado en la Figura 9.2, tanto las instancias que forman parte de la arquitectura, como la topología de red de la arquitectura, son administrados por un mismo servidor con Xen Project para la gestión de ambientes virtualizados con las especificaciones mostradas en la Tabla 9.1. La mayoría de los resultados de las pruebas son limitadas hasta esos rangos por la incapacidad a nivel de procesamiento con la arquitectura mencionada.

En aras de mejorar los resultados encontrados en algunas de las pruebas de estrés realizadas, el hipervisor Xen Project permite la gestión de su arquitectura de compartición de recursos, principalmente a nivel de CPU [29]. La importancia de esto se debe a que no todas las instancias que corren dentro del mismo servidor, tienen los mismos requerimientos a nivel de recursos de CPU. Por ello, la configuración por defecto provista por el hipervisor Xen y la utilización de un único vCPU para cada

instancia dentro del hipervisor, conducía incluso a peores resultados. Múltiples intentos de distintas distribuciones de los recursos fueron intentados, desde crear afinidad entre núcleos de los CPU con los vCPUs de algunas instancias hasta distribución de prioridad entre las distintas instancias, pero aquellas que obtuvieron mejores resultados se pueden ver en la Tabla 11.4.

Máquinas	Gestión de vCPUs	Gestión de Peso (Schedule-Credit)
Domain-0	4 vCPU, ninguna afinidad	256
tegamjg_db1	1 vCPU, ninguna afinidad	512
tegamjg_pfsense1	1 vCPU, ninguna afinidad	512
tegamjg_pfsense2	1 vCPU, ninguna afinidad	512
tegamjg_pbx1	2 vCPU, ninguna afinidad	1000
tegamjg_pbx2	2 vCPU, ninguna afinidad	1000
tegamjg_kam1	2 vCPU, ninguna afinidad	1000
tegamjg_kam2	2 vCPU, ninguna afinidad	1000

Tabla 11.4: Gestión de Compartición de Recursos en la Arquitectura

Algunos inconvenientes con la herramienta SIPp fueron presenciados al momento de correr las pruebas debido a su deficiencia en la realización de escenarios con gestión de tráfico RTP. Estas limitaciones se basan en que SIPp tiene una arquitectura basada en un único hilo donde los eventos son repetidos mediante un bucle dentro del mismo hilo [25], lo que produce que los archivos .pcap descritos en la Sección 6.6.2 y 9.4.6 sean gestionados por un mismo núcleo del procesador de la maquina con SIPp. Así, al momento de llegar a un máximo de 800 llamadas aproximadamente activas, el núcleo del procesador llegaba a un 100% de su capacidad, generando así incapacidad por parte del escenario de SIPp de funcionar correctamente. Por ello, se opta por realizar aquellos escenarios descritos en la Sección 9.4.6.1 y 10.3.2 de la misma manera pero sin la gestión de tráfico RTP.

11.2.1 Prueba de Flujo de Mensajes INVITE entre dos Endpoints

Las pruebas de estrés realizadas para mensajes INVITE tuvieron resultados variados, principalmente por capacidades a nivel de recursos (mayormente a causa de limitaciones de CPU) y, en ciertas pruebas, la decisión de pasar el tráfico RTP a través de los nodos de Asterisk por funcionalidades como transcoding o grabación de llamadas como lo muestra la Tabla 9.3 y Figura 9.8. Las principales razones se basan en el consumo de recursos de CPU debido a la apertura de sesiones RTP y SIP, lo que conlleva a la creación de sockets e hilos para permitir las retransmisiones de tráfico RTP entre los endpoints que forman parte de las llamadas. Para el escenario de prueba 1, se realizaron las pruebas descritas en la Tabla 10.4 que describen el Escenario 1. Los resultados de las mismas se pueden ver en la Tabla 11.5.

Prueba	Duración	Promedio de Tasa de Llamadas	Llamadas Generadas	Promedio de Llamadas Activas	Llamadas Exitosas	Llamadas Fallidas	Retransmisiones
Prueba 1	1 hora	284.75	1025107	1997.583	1025091	16	0

		284.807	1025314	1997.667	1024246	1068	34
Prueba 2	1 hora	249.343	897640	1997.667	897626	14	13
		248.771	897573	1997.75	897562	11	58
Prueba 3	20 minutos	345.767	418292	344.2	53614	364574	29308
		346.875	420175	268	59533	360638	33832
Prueba 4	1 hora	284.877	1025566	1998.333	1025001	565	0
		284.763	1025162	1998.792	1025143	19	4

Tabla 11.5: Resultados de las Pruebas de Estrés Realizadas por mensajes INVITE en el Escenario 1

Entre estas pruebas realizadas, se verifica la capacidad de escalar llamadas entre múltiples servidores con Asterisk mediante mecanismos de balanceo de carga. La Figura 11.2 muestra la cantidad de llamadas activas en ese momento y la cantidad de llamadas procesadas para la Prueba 1 descrita en la Tabla 11.5. Es importante destacar que para las pruebas realizadas, en Kamailio se configura un máximo de carga de 1000 llamadas activas por cada nodo de Asterisk. Esto debido a que se pudo presenciar un cope de CPU del 100% una vez establecidas alrededor de esa cantidad de llamadas.

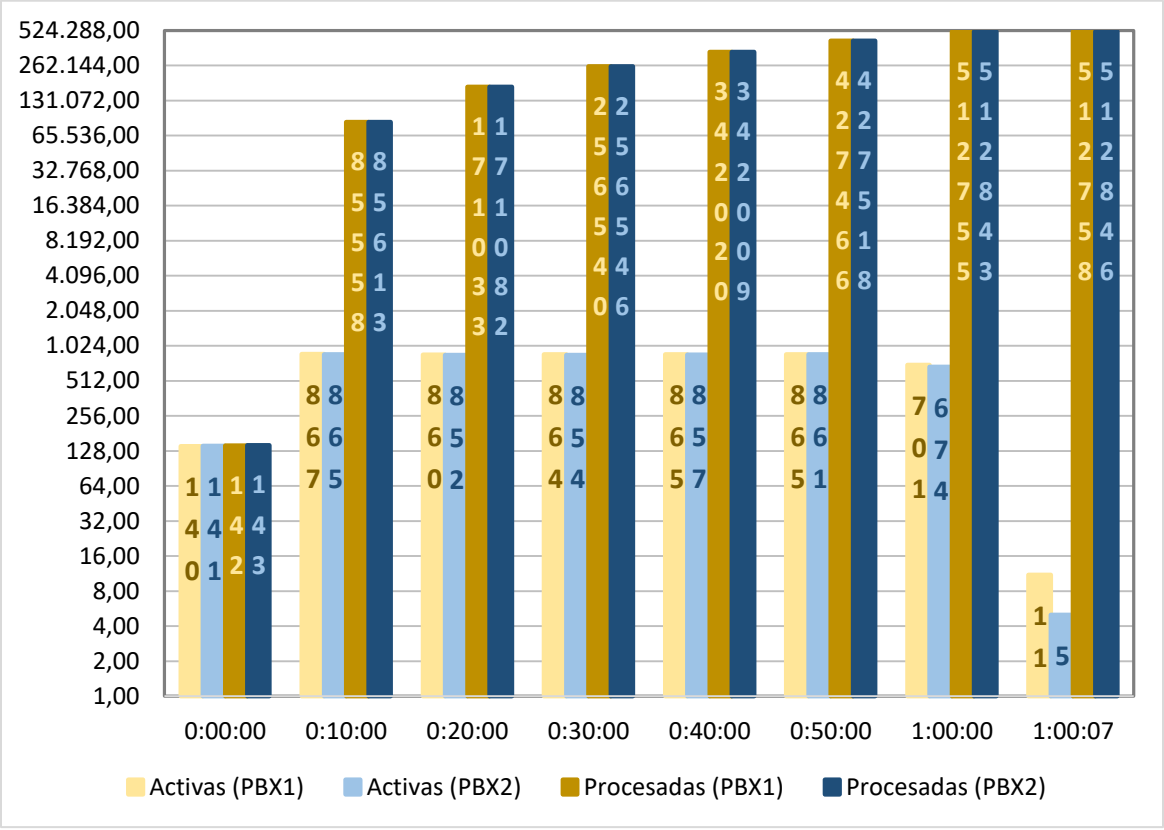


Figura 11.2: Gráfica demostrando la Gestión de Llamadas por parte de los Nodos con Asterisk

```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --^M
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host^M
 300.0(0 ms)/1.000s   5061    3607.03 s      1025107  10.0.1.207:5060(UDP)^M
^M
  0 new calls during 0.000 s period      0 ms scheduler resolution^M
  0 calls (limit 2000)                   Peak was 2000 calls, after 6 s^M
  0 Running, 9402 Paused, 0 Woken up^M
  16 dead call msg (discarded)           0 out-of-call msg (discarded)        ^M
  1 open sockets                        ^M
^M
                                Messages  Retrans   Timeout   Unexpected-Msg^M
      INVITE ---------->        1025107   0         0                 ^M
         100 <----------        0         0         0         0       ^M
         180 <----------        0         0         0         0       ^M
         407 <----------        1025107   0         0         0       ^M
         ACK ---------->        1025107   0                           ^M
      INVITE ---------->        1025107   0         0                 ^M
         100 <----------        1025107   0         0         0       ^M
         180 <----------        1014798   0         0         0       ^M
         200 <----------  E-RTD1 1025107   0         0         0       ^M
         ACK ---------->        1025107   0                           ^M
^M
         100 <----------        0         0         0         0       ^M
         180 <----------        0         0         0         0       ^M
         200 <----------  E-RTD1 0         0         0         0       ^M
         ACK ---------->        0         0                           ^M
^M
       Pause [   6000ms]        1025107                       16       ^M
         BYE ---------->        1025091   0         0                 ^M
         200 <----------  E-RTD1 1025091   0         0         0       ^M
       Pause [   1000ms]        1025091                       0        ^M
------------------------------ Test Terminated --------------------------------^M
```

Figura 11.3: Flujo del Evento de SIPp y Resultados de la Prueba 1

A diferencia de las fallas en las llamadas de la Prueba 3 y algunas otras, Las fallas en las otras pruebas no son consideradas fallas como tales, y ello principalmente se debe a la recepción primaria del mensaje de respuesta 200 OK para iniciar la sesión INVITE antes de la recepción del mensaje 180 Ringing. Eso implica que, de igual manera, la llamada es creada correctamente, pero debido a esa desincronización en la recepción del mensaje, la misma no es terminada por parte de SIPp. Otras de las posibles fallas (Prueba 1, numero 2 y Prueba 4, numero 1) se debían a cuestiones de procesamiento, donde Asterisk está en la capacidad de analizar y definir un estado de congestión (Dialstatus: CONGESTION) dentro del servidor al inicio de alguna llamada, respondiendo con un mensaje 500 al inicio de sesión INVITE de algún usuario. No se consiguió una fuente confiable que explique las razones, de manera más profunda, por la cual Asterisk puede asumir encontrarse en un estado de congestión, claramente, en este caso, se debe al alto consumo de CPU, pero no se explica con mayor claridad.

Ya comentadas, al inicio de esta sección, algunas de las limitaciones en las pruebas realizadas tienen que ver con la capacidad a nivel de procesamiento por las causas ya explicadas. Estas limitaciones en los resultados de las pruebas pueden verse claramente en la Prueba 3. En la mayoría de estas pruebas, se pudo apreciar que los momentos donde la arquitectura sufría mayores inconvenientes se ubican en los primeros minutos de dichas pruebas, donde, en la Prueba 3, la arquitectura no se bastó de la carga, por cuestiones de procesamiento, para culminar con la prueba de manera correcta.

En condiciones de alta carga de procesamiento, los esquemas de clustering con Pacemaker/Corosync pueden presentar pérdidas en su comunicación, y en ciertos casos, disminuir la recuperación de la gestión del mismo en caso de alta carga, mediante el parámetro load-threshold del demonio CRMd que por defecto es de 80%. La Figura 11.4 muestra la falla en la sincronización de los tokens por parte de Corosync, lo que conlleva al mismo a tratar de recuperar y reformar la infraestructura formada anteriormente.

```
10: Sep 14 17:23:13 [3431] pbx1       crmd:   notice: throttle_handle_load:High CPU
    load detected: 12.730000
11: [3409] pbx1 corosyncwarning [MAIN  ] Corosync main process was not scheduled for
    1053.9088 ms (threshold is 800.0000 ms). Consider token timeout increase.
12: [3409] pbx1 corosyncnotice  [TOTEM ] A processor failed, forming new
    configuration.
13: [3409] pbx1 corosyncnotice  [TOTEM ] A new membership (10.0.3.200:3228) was
    formed. Members
14: [3409] pbx1 corosyncwarning [MAIN  ] Corosync main process was not scheduled for
    923.3477 ms (threshold is 800.0000 ms). Consider token timeout increase.
15: [3409] pbx1 corosyncnotice  [QUORUM] Members[2]: 1 2
16: [3409] pbx1 corosyncnotice  [MAIN  ] Completed service synchronization, ready to
    provide service.
17: Sep 14 17:23:17 [3425] pbx1 pacemakerd:     info: pcmk_quorum_notification:
    Membership 3228: quorum retained (2)
18: Sep 14 17:23:17 [3431] pbx1       crmd:     info: pcmk_quorum_notification:
    Membership 3228: quorum retained (2)
```

Figura 11.4: Falla de Sincronización en el Esquema de Clustering de Componentes PBX

Estas fallas de sincronización causan comportamientos anormales dentro del esquema de clustering que conducen a fallas en distintos recursos gestionados por el esquema, que conllevan, en ciertos casos, a medidas drásticas como mecanismos de fencing. En este caso en particular, esa falla de sincronización conlleva al error en el monitoreo de Asterisk que luego, al intentar reiniciar el servicio de Asterisk, el mismo no logra parar, lo que conduce, basándose en las políticas configuradas, a la aplicación de fencing a ese nodo.

Para el escenario de prueba 2, se realizaron las pruebas descritas en la Tabla 10.4 que describen el Escenario 2. Los resultados de las mismas se pueden ver en la Tabla 11.6.

Prueba	Duración	Promedio de Tasa de Llamadas	Llamadas Generadas	Promedio de Llamadas Activas	Llamadas Exitosas	Llamadas Fallidas	Retransmisiones
Prueba 1	1 hora	142.424	512733	998.3333	512730	3	0
		142.424	512731	998.375	512726	5	0
Prueba 2	1 hora	166.002	597613	997.7917	512730	3	0
		165.979	597530	997.7917	597522	8	0
Prueba 3	1 hora	197.705	307064	509.2727	98073	207796	23530
		199.102	717763	1197.625	717658	105	104
Prueba 4	1 hora	170.863	615960	1199.042	615859	101	0
		170.856	615953	1198.833	615830	123	2

Tabla 11.6: Resultados de las Pruebas de Estrés Realizadas por Mensajes INVITE en el Escenario 2

Entre las primeras impresiones de los resultados de las pruebas realizadas en el Escenario 2, destacan que los mismos no demuestran un comportamiento, teóricamente “esperado”. Todo esto debido a que, en conclusión, las pruebas en este escenario no indican un soporte del 50% de la carga utilizada para la medición del Escenario 1, sino un poco mayor. La respuesta a esta incógnita es razonable si se toma en cuenta que todas las instancias son virtualizadas por un mismo servidor, y la distribución en la gestión de los recursos es totalmente compartida. La afinidad de núcleos de CPU con los vCPUs de las máquinas virtualizadas fue anteriormente intentado, pero conllevaban a peores resultados, quizás porque la cantidad de vCPUs requeridos en total es mucho mayor a la cantidad de núcleos físicos, lo que podría conducir a tiempos de inutilización de algún núcleo por más tiempo por parte de las máquinas virtuales que forman parte de la arquitectura. Además, al encontrarse un único nodo de Asterisk dentro del esquema de clustering, aquel fenómeno en la comunicación de Corosync descrito en la Figura 11.4, no ocurre debido a que no requiere sincronizarse con ningún otro nodo dentro del esquema de clustering.

Es importante destacar que las Pruebas 3 y 4 permiten a Asterisk tener un total de 1200 como máxima cantidad de llamadas, en vez de las 1000 utilizadas en el Escenario 1, debido a que el comportamiento de las pruebas mostraba buenos resultados. Es importante destacar que, a nivel de procesamiento, 1000 llamadas ya mostraban una alta carga, por ello en la Prueba 3 número 1 se puede observar que las pruebas con un máximo de 1200 es variable en sus buenos resultados, debido a que el recurso de Asterisk, en su gestión de monitoreo, falló un total de 2 veces así llegando de tal modo el contador de falla a 2 e inhabilitando proveer el servicio de vuelta.

Para el escenario de prueba 3, se realizaron las pruebas descritas en la Tabla 10.4 que describen el Escenario 3. El resultado de las mismas se puede ver en la Tabla 11.7.

Prueba	Duración	Promedio de Tasa de Llamadas	Llamadas Generadas	Promedio de Llamadas Activas	Llamadas Exitosas	Llamadas Fallidas	Retransmisiones
Prueba 1	1 hora	298.865	1075921	423.4348	144530	931391	64355
		296.284	1066628	409.4167	179795	886833	71324
Prueba 2	1 hora	227.199	817925	1597.542	817924	1	4
		227.057	816506	1597.201	816506	0	0
Prueba 3	1 hora	247.908	892480	1992.083	892472	8	44
		247.92	892523	1991.375	892521	2	24
Prueba 4	1 hora	248.006	892833	1995.833	892831	2	679
		248.357	894094	1997.292	894085	9	261

Tabla 11.7: Resultados de las Pruebas de Estrés Realizadas por Mensajes INVITE en el Escenario 3

A diferencia de las pruebas realizadas en el Escenario 1, donde se cuenta igual con ambos servidores de Asterisk, el proxy RTP que forma parte del nodo de Kamailio se encarga de la retransmisión del tráfico por la red pública hacia el endpoint que se encuentra fuera de la arquitectura de red de la implementación. Esto conlleva a un considerable aumento en el procesamiento que sufre en general la arquitectura por la política de gestión de compartición de recursos de CPU de Xen Project.

Aunque el escenario de SIPp utilizado en las pruebas del mismo es prácticamente igual al utilizado en los otros dos, es relevante destacar que, por la naturaleza del proxy RTP de actuar como un proxy simétrico, es necesaria la recepción de tráfico RTP por ambos endpoints para establecer la retransmisión correcta del proxy del tráfico RTP. Por ello, se incluye el envío de un tono DTMF entre ambos endpoints para la realización de la prueba.

Por el conocimiento previo de las pruebas realizadas en el Escenario 1, se optó por comenzar con la Prueba 1 del Escenario 1 en el Escenario 3. Lo comentado en el párrafo anterior se vio reflejado en el resultado de la prueba, donde la causa y el comportamiento de dichas fallas fueron prácticamente idénticas a los presentados en la Prueba 3 del Escenario 1. Ahora, a diferencia de esas pruebas, la Prueba 1 se dejó toda la hora completa, lo cual se observa cómo se comporta el parámetro de **failout-timeout** configurado en el esquema de clustering de Asterisk. En la Figura 11.5 se puede observar cómo se comporta la prueba realizada en la Prueba 1 numero 1.

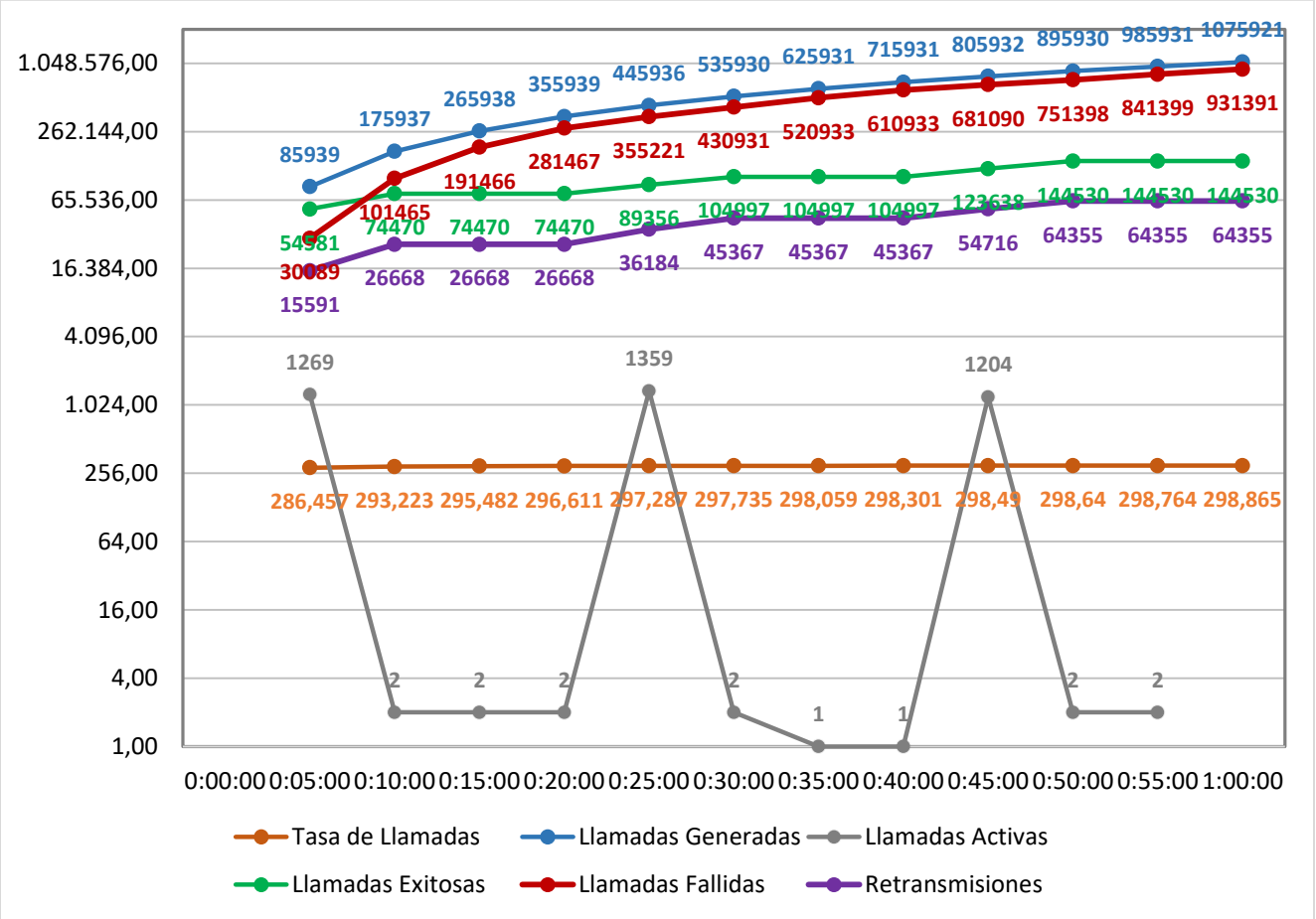


Figura 11.5: Gráfica Relacionada a los Resultados Obtenidos en la Prueba 1 Numero 1 en el Escenario 3

En la Figura 11.5 se observa como luego de pasar 10-12 minutos entre la falla de ambos nodos de Asterisk, el contador de falla del servicio de Asterisk es reseteado para permitir nuevamente el inicio del servicio de Asterisk (en este caso, solamente en uno de los nodos debido que en el otro fue aplicado fencing al producirse la primera falla) como se puede ver entre los intervalos de 20 a 30 minutos y de 40 a 50 minutos de haber realizado la prueba.

Algunos inconvenientes se presentaron al realizar las pruebas para este escenario. Entre ellas se encontraron que cualquiera de las 4 pruebas realizadas producían que el recurso de Kamailio fallará en el proceso de monitoreo múltiples veces a lo largo de las distintas pruebas, lo que conlleva a la migración del servicio al nodo pasivo y, en ese intervalo, se producían fallas en las llamadas realizadas. Esto fue percibido como un comportamiento inesperado a lo largo de las pruebas realizadas. Para nuestra sorpresa, se percibe que la función de monitoreo del recurso de Kamailio mediante el envío de mensajes OPTION con SIPSAK no posee mecanismos de retransmisión internos por parte de SIPSAK al no recibir respuesta inmediata al mensaje OPTION, en las pruebas de este escenario, donde la carga a nivel de procesamiento aumenta en el nodo en el que se encuentra Kamailio siendo gestionado, se especifica la capacidad de proveer mecanismos que permitan el reenvío de mensajes OPTION hasta recibir una

respuesta correcta por parte del nodo en un pequeño intervalo de tiempo como una modificación ligera al recurso OCF de Kamailio explicado en la Sección 9.4.5.3.

11.2.2 Prueba de Flujo de Mensajes REGISTER

Las pruebas de estrés realizadas para mensajes REGISTER tuvieron resultados variados aunque un poco mejores que en los mensajes INVITE. En este caso el Dominio 0, Firewall de la arquitectura, la base de datos (los tres reciben mayor carga que en la prueba anterior) y Kamailio, son los principales componentes que son afectados por parte de esta prueba.

Para el escenario de prueba 1, se realizaron las pruebas reseñadas en la Tabla 10.5 que describen el Escenario 1. El resultado de las mismas se puede ver en la Tabla 11.8.

Prueba	Duración	Promedio de Tasa de Transacciones	Transacciones Generadas	Transacciones Exitosas	Transacciones Fallidas	Retransmisiones
Prueba 1	1 hora	498.812	1795733	1795733	0	72855
		498.866	1795932	1795932	0	71952
Prueba 2	1 hora	748.159	2693403	2693385	18	232129
		748.292	2693882	2693882	0	174833
Prueba 3	1 hora	996.051	3585832	3585831	1	301450
		995.236	3582895	3582887	8	275637

Tabla 11.8: Resultados de las Pruebas de Estrés Realizadas por Mensajes REGISTER en el Escenario 1

Para el escenario de prueba 3, se realizaron las pruebas descritas en la Tabla 10.5 que describen el Escenario 3. El resultado de las mismas se puede ver en la Tabla 11.9.

Prueba	Duración	Promedio de Tasa de Transacciones	Transacciones Generadas	Transacciones Exitosas	Transacciones Fallidas	Retransmisiones
Prueba 1	1 hora	498.061	1793033	1793033	0	93442
		498.222	1793613	1793613	0	95194
Prueba 2	1 hora	748.377	2694187	2694187	0	189374
		748.01	2692861	2692861	0	218301
Prueba 3	1 hora	994.551	3580431	3580426	5	380665
		993.666	3577243	3577242	1	287888

Tabla 11.9: Resultados de las Pruebas de Estrés Realizadas por Mensajes REGISTER en el Escenario 3

Para las pruebas realizadas en ambos escenarios, se percibieron comportamientos parecidos. Dichos comportamientos consisten en la culminación de las transacciones REGISTER de manera exitosa aunque con altas retransmisiones, para el Escenario 3 en particular las retransmisiones son un poco más altas. Es importante destacar que en este caso se registra un total de 10000 usuarios anteriormente no registrados en la arquitectura, luego de las primeras 10000 peticiones de registros el resto de esas

transacciones son utilizadas para actualizar el estado del registro de dicho usuario de manera cíclica hasta culminar la prueba.

En conclusión a este capítulo, se logra determinar un buen comportamiento relacionado a los mecanismos de contingencia, que permiten un alto nivel de transparencia entre las caídas del servicio y su próxima recuperación casi inmediata. Son en las pruebas de estrés donde los resultados fueron algo variados, principalmente por razones a nivel de consumo de recursos físicos. Sin embargo, los mecanismos de balanceo de carga agregan una distribución prácticamente equitativa del tráfico a los distintos nodos de Asterisk, con capacidad en el control de flujo por cantidad de llamadas activas en cada uno de dichos nodos.

Conclusiones y Futuros Trabajos

VoIP es una tecnología que ha tenido un gran impacto en el área de la comunicación mediante el uso de redes digitales, estas han servido como plataforma para el desarrollo de una gran variedad de herramientas tanto propietarias como de software libre, tal es el caso particular de Asterisk, herramienta en la cual la comunidad ha descrito un desarrollo incremental enfocado en la inclusión y desarrollo de múltiples servicios y soporte a múltiples protocolos de señalización para lograr proveer un servicio de telefonía de calidad por sí solo, el cual ha sacrificado aspectos de escalabilidad, gestión eficiente de ciertos y mecanismos de contingencia en caso de fallas.

Es por ello que en el presente trabajo especial de grado el uso de una metodología de trabajo orientada a la división de servicios para el máximo aprovechamiento de las cualidades de cada herramienta de software seleccionada representó un punto de éxito, ya que si bien una herramienta puede tener ciertas bondades también es posible que tenga puntos débiles, aquí es donde la metodología de división de servicios describe su máximo potencial, permitiendo fortalecer las debilidades en algún área de una herramienta mediante la utilización de otra en la misma área..

Adicionalmente en otros aspectos durante el desarrollo de la arquitectura de red se pudieron evidenciar que la buena utilización de los conceptos de alta disponibilidad, balanceo de carga y mecanismos de contingencia permiten de manera efectiva la construcción de arquitecturas de red que prioricen la continuidad de la operación y control y buena gestión de servicios altamente disponibles.

En cuanto al flujo de comunicación el uso del protocolo SIP cumple un papel importante ya que el mismo describe patrones de comunicación que lo hacen un protocolo ligero permitiendo elevados flujos de comunicación y alta concurrencia.

Empleando el hipervisor XenProject se comprobó que la utilización de arquitecturas virtualizadas permite la construcción de plataformas de comunicación escalables y robustas que prioricen la continuidad de la operación al mismo tiempo que permitan la aplicación de mecanismos de contingencia para la recuperación ante fallos o caídas del servicio, sin embargo para la aplicación de mecanismos de fencing dentro del hipervisor se pudo evidenciar cierta debilidad la cual podría ser mitigada con mecanismos de fencing por hardware.

La gestión inteligente de recursos virtualizados y la realización de pruebas de estrés sobre la arquitectura de red permite dimensionar el tráfico soportado así como también definir mecanismos de control de flujo que funcionan como parámetros garantes de la operatividad de la arquitectura en casos alto consumo de servicios.

Finalmente se puede concluir que los objetivos planteados como puntos focales de la arquitectura de red fueron cumplidos a cabalidad, pudiéndose generar una arquitectura de red que garantiza la alta disponibilidad de recursos y prioriza la continuidad de la operación permitiendo asegurar el correcto funcionamiento de la red telefonía para centrales telefónicas basadas en VoIP.

Trabajos Futuros

Distintas soluciones podrían ser implantadas para proveer dichos mecanismos anteriormente nombrados para la implementación de una arquitectura de alta disponibilidad de telefonía VoIP. Es por ello que, como trabajos futuros, se propone la implementación de una arquitectura de alta disponibilidad de telefonía VoIP basadas en el protocolo SIP, con la utilización de herramientas de servidores SIP como Kamailio u OpenSER, en conjunto con conceptos de esquema de clustering con herramientas como Pacemaker, Corosync, CMAN, entre otros, para ofrecer los mecanismos anteriormente nombrados con una óptica de balanceo de carga relacionada a los servicios de telefonía deseados a proveer. Como ejemplo propuesto se tendría un conjunto de servidores Asterisk, FreeSWITCH, entre otros, donde cada uno se encargaría de proveer distintos servicios de telefonía, como Voicemail, Conferencing, ACL y afines, ofreciendo mecanismos de contingencia para dichos servicios.

Referencias Bibliográficas

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, SIP: Session Initiation Protocol, RFC 3261, June 2002

[2] P. Resnick, Internet Message Format, RFC 2822, April 2001.

[3] A. Niemi, Session Initiation Protocol (SIP) Extension for Event State Publication, RFC 3903, October 2004.

[4] J. Rosenberg, The Session Initiation Protocol (SIP) UPDATE Method, RFC 3311, September 2002.

[5] R. Sparks, The Session Initiation Protocol (SIP) Refer Method, RFC 3515, April 2003.

[6] A. R. Tekelec, SIP-Specific Event Notification, June 2012.

[7] C. Gonzalo, SIP Demystified, New York: McGraw-Hill, 2002.

[8] P. Weygant, Cluster for High Availability: A Primer HP Solution, 2nd Edition, Prentice Hall, 2001.

[9] E. Marcus, Blueprints for High Availability, 2nd Edition, Wiley Publishing, September 2003.

[10] M. Resman, CentOS High Availability, Packt Publishing, April 2015.

[11] C. Kopparapu, Load Balancing, Servers, Firewalls and Caches, Wiley Computer Publishing, 2002.

[12] H. Jiang et al, Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters, Huazhong University of Science and Technology, 2012.

[13] K. Singh and H. Schulzrinne, Failover and Load Sharing in SIP Telephony, Department of Computer Science, Columbia University, 2004.

[14] W. Tarreau, Making Application Scalable With Load Balancing, September 2006.

[15] Load Balance Voice Over IP SIP Traffic in your BladeCenter Economically and Efficiently with the Layer 2-7 Gigabit Ethernet Switch Modules from BLADE Network Technologies, BLADE Network Technologies, 2009.

[16] F. Goncalves, Configuration Guide for Asterisk PBX, V.Office Networks, March 2007.

[17] R. Bryant, L. Madsen and J. Van Meggelen, Asterisk: The Definitve Guide, 4th Edition, O'Reilly Media, 2013.

[18] N. Anaya, Fundamentos de Telefonía IP e Introducción a Asterisk/Elastix, ElastixTech, 2013.

[19] C. J. J. Peterson, Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP), August 2006.

[20] S. Dake, C. Caulfield and A. Beekhof, The Corosync Cluster Engine, in Proceedings of the Linux Symposium, Vol. 1, Ottawa, Ontario, Canada, pp. 85-100, July 2008,.

[21] A. Beekhof, Pacemaker: Configuration Explained 1.1, 5th Edition, 2015.

[22] P. Reisner and L. Ellenberg, DRBD: The User's Guide, LINBIT Information Technologies GmbH, February 2011.

[23] R. Chase, OCFS2 Best Practices Guide: An Oracle White Paper, Oracle Corporation, February 2014.

[24] S. Whitehouse, The GFS2 Filesystem, In the Proceedings of The Linux Symposium, Vol. 2, pp. 253-260, July 2007.

[25] 0. Jacques and R. Day, SIPp Reference Documentation, 2013.

[26] G. Kambourakis el at, High Availability for SIP: Solutions and Real-Time Measurement Performance Evaluation, International Journal of Disaster Recovery and Business Continuity, Vol. 1, No. 1, pp. 11-29, February 2010.

[27] S. Pal, R. Gadde, and H Latchman, On the Reliability of Voice Over IP (VoIP) Telephony, University of Florida, 2011.

[28] M. Pohančeník, Diploma Thesis: Design and Implementation of a System to Interconnect VoIP Services and CERN’s Telephony Network, University of Žilina,

Ginebra, Suiza, April 2013.

[29] L. Crawford, C. Takemura, The Book of Xen: A Practical Guide for the System Administrator,San Francisco, United States of America, No Starch Press Inc., 2010.