PROYECTO FINAL DE CARRERA

IMPLEMENTACIÓN DE UNA PLATAFORMA IMS CON HERRAMIENTAS OPEN SOURCE

Estudios: Ingeniería de Telecomunicaciones
Autor: Margarita Garrido Lorenzo
Director/a: José Luis Muñoz Tapia
Año: 2016

Índice general

Capítulo 1

Resumen

Históricamente, las llamadas de voz y otros servicios de telefonía se han proporcionado a través de una red de conmutación de circuitos. Sin embargo, la implementación, despliegue y evolución de servicios en redes basadas en circuitos, tanto fijas como móviles, es una labor tediosa mientras que estas mismas operaciones en las redes de datos IP se hacen de forma más ágil, flexible y unificada con otros servicios.

Más aún, hoy en día, el concepto de telefonía se ha enriquecido con nuevos servicios y ya no se asocia exclusivamente con las llamadas de voz. Es por ello que los operadores y proveedores de servicios en general ven la necesidad de evolucionar hacia un modelo que les permita proveer y desplegar de forma ágil nuevos servicios para satisfacer la demanda de sus usuarios.

De esta necesidad nace IMS, creando un marco arquitectónico capaz de proveer servicios multimedia en tiempo real, como sesiones de voz, vídeo y conferencia, y servicios que no requieren de tiempo real, como presencia o mensajería instantánea. Todo ello sobre infraestructura IP que permite una integración horizontal de servicios, es decir, diversos servicios sobre una única red, con las ventajas que ello conlleva respecto a la operación y mantenimiento. Cabe destacar que la integración horizontal es un concepto clave en los estándares de redes de nueva generación o NGN (Next Generation Networks) propuestos por el ITU como evolución de las redes clásicas.

El objetivo principal de este Proyecto Fin de Carrera es el estudio práctico de las funciones principales de un sistema IMS y servicios añadidos, presencia y buzón de voz, mediante su implementación con herramientas de código abierto. Pero no se puede entender una arquitectura sin conocer los protocolos que ésta utiliza. Es por ello que este proyecto se inicia con el estudio de SIP, protocolo por excelencia para la gestión y establecimiento de sesiones multimedia.

Así, la primera parte se centra no sólo en el estudio del protocolo, si no también en su interacción con otros servicios y protocolos que lo harán evolucionar y enriquecerán y sin los cuales no se puede entender el IMS.

Los dos grandes bloques de este proyecto, SIP e IMS, se han estructurado de forma similar. Una primera parte en la que se hace un estudio teórico y una segunda parte en la que se analiza y profundiza en los conceptos teóricos a través del desarrollo de prácticas. Las prácticas analizan implementaciones reales mediante un entorno de nodos virtualizados.

En el caso de IMS, se ha implementado un sistema con las funciones P-CSCF, I-CSCF, S-CSCF, HSS y dos AS, uno para presencia y otro como buzón de voz. Todo ello mediante el uso de las herramientas `pjsua`, `bind9`, `FHoSS`, `Kamailio` y `Asterisk`, todas ellas de código abierto. Las prácticas de este bloque se ha diseñado para proporcionar una amplia y detallada visión de lo que podría ser un IMS real con las funcionalidades mencionadas.

Capítulo 2

Resum

Històricament, les trucades de veu i altres serveis de telefonia s'han proporcionat mitjançant xarxes de commutació de circuits. No obstant això, la implementació, desplegament i evolució de serveis en xarxes basades en circuits, tant fixes com mòbils, és una tasca tediosa mentre que aquestes mateixes operacions en les xarxes de dades IP es fan de forma més àgil, flexible i unificada amb altres serveis.

Més encara, avui dia, el concepte de telefonia s'ha enriquit amb nous serveis i ja no sássocia exclusivament amb les trucades de veu. És per això que els operadors i proveïdors de serveis en general veuen la necessitat dévolucionar cap a un model que els permeti proveir i desplegar de forma àgil nous serveis per satisfer la demanda dels seus usuaris.

Dáquesta necessitat neix IMS, creant un marc arquitectònic capaç de proveir serveis multimèdia en temps real, com sessions de veu, vídeo i conferència, i serveis que no requereixen de temps real, com presència o missatgeria instantània. Tot això sobre infraestructura IP que permet una integració horitzontal de serveis, és a dir, diversos serveis sobre una única xarxa, amb els avantatges que això comporta pel que fa a lóperació i manteniment. Cal destacar que la integració horitzontal és un concepte clau en els estàndards de xarxes de nova generació o NGN (Next Generation Networks) proposats per lÍTU com a evolució de les xarxes clàssiques.

Lóbjectiu principal dáquest Projecte Fi de Carrera és léstudi pràctic de les funcions principals dún sistema IMS i serveis afegits, presència i bústia de veu, mitjançant la seva implementació amb eines de codi obert. Però no es pot entendre una arquitectura sense conèixer els protocols que aquesta utilitza. És per això que aquest projecte sínicia amb léstudi de SIP, protocol per excel·lència per a la gestió i establiment de sessions multimèdia.

Així, la primera part se centra no només en léstudi del protocol, sinó també en la seva interacció amb altres serveis i protocols que ho faran evolucionar i enriquiran i sense els quals no es pot entendre lÍMS.

Els dos grans blocs dáquest projecte, SIP i IMS, s'han estructurat de manera similar. Una primera part en la qual es fa un estudi teòric i una segona part en la qual sánalitza i sáprofundeix en els conceptes teòrics a través del desenvolupament de pràctiques. Les pràctiques sánalitzen implementacions reals mitjançant un entorn de nodes virtualitzats.

En el cas dÍMS, s'ha implementat un sistema amb les funcions P-CSCF, I-CSCF, S-CSCF, HSS i dues AS, un per presència i un altre com bústia de veu. Tot això mitjançant l'ús de les eines `pjsua`, `bind9`, `FHoSS`, `Kamailio` i `Asterisk`, totes elles de codi obert. Les pràctiques dáquest bloc s'ha dissenyat per proporcionar una àmplia i detallada visió del que podria ser un IMS real amb les funcionalitats esmentades.

Capítulo 3

Abstract

Historically, telephony and other voice services have been provided through circuit-switched networks. However, the implementation, deployment and evolution of services in these kind of networks, both fixed and mobile, is a tedious work while these same operations become more agile, flexible and unified with other services in IP data networks.

Moreover, today, the concept of telephony has been enriched with new services and is no longer only associated with voice calls. That is why operators and service providers began to see the need to move towards a model that allows them to provide and deploy new services with flexibility in order to meet the demand of users.

The working group of IMS emerges from this need, creating an architectural framework capable of providing real-time multimedia services such as voice sessions, video and conferencing, and services that do not require real time, as presence or instant messaging. It is designed to operate over IP infrastructure which enables a horizontal integration of services, ie various services over a single network, with the advantages that this entails regarding operation and maintenance. Note that the horizontal integration is a key concept in the standards of NGN (Next Generation Networks) proposed by the ITU as an evolution of traditional networks.

The main objective of this Thesis is the practical study of the main functions of an IMS system and value-added services, presence and voicemail, through its implementation with open source tools.

The understanding of an architecture requires the understanding of the protocols that it uses. That is why this Thesis begins with the study of SIP, a widely adopted protocol which purpose is the management and establishment of multimedia sessions and on which IMS is based.

Thereby, the first part focuses not only on the study of this protocol, but also in its interaction with other services and protocols that will enrich it and without which IMS can not be conceived.

The two main blocks, SIP and IMS, are structured similarly. There is a first part in which we focus on a theoretical study and a second one that analyzes and delves into those theoretical concepts through the development of practices. These practices analyze actual scenarios using virtualized environments based on vnuml.

What stands of the IMS block is the implementation made of a system with the main functions P-CSCF, I-CSCF, S-CSCF, HSS and two AS, one for presence and another for voicemail. All this by using open source tools as *pjsua*, *bind9*, *FHoSS*, *Kamailio* and *Asterisk*. The practices of this block are designed to provide a comprehensive and detailed vision of what could be a real IMS with the features mentioned.

Capítulo 4

Introducción

4.1. Contexto del proyecto

IMS nace de la necesidad de proveer servicios, tanto en tiempo real como no tiempo real, con calidad de servicio y sobre una infraestructura que permita integración horizontal.

Pero, ¿por qué IMS y no otra solución VoIP?

El principal problema de las redes de conmutación de paquetes es que proporcionan, por lo general, servicios multimedia en tiempo real sin calidad de servicio. La red no ofrece ninguna garantía sobre la cantidad de ancho de banda que puede disponer un usuario o de los retrasos en la entrega de los paquetes. En consecuencia, la calidad de una conversación VoIP puede variar dramáticamente a lo largo de su duración. Una de las razones para crear el IMS fue disponer de la capacidad para proporcionar QoS.

La prestación de servicios integrados para los usuarios es la otra razón principal de la existencia del IMS. Los operadores quieren ser capaces de utilizar los servicios desarrollados por terceros, combinarlos, integrarlos con los servicios que ya tienen, y proporcionar al usuario un servicio completamente nuevo.

La tercera gran razón la encontramos en las redes de telefonía móvil. IMS permite a los operadores determinar en todo momento en que contexto se sitúa el tráfico de datos generados por los usuarios, si es una videoconferencia, una llamada VoIP o navegación web por ejemplo. Esto permite disponer de un sistema de tarificación adecuado y un sistema de reparto de carga basado en el tipo de flujo.

Por todo ello, IMS está pisando fuerte en el mundo de la VoIP y especialmente en VoLTE (Voz Sobre LTE). Tanto es así, que, según la firma IHS, los ingresos del mundo IMS y VoIP registró el año pasado un aumento del 31 % situándose en 5.5 mil millones de dólares a finales de 2015, el mejor año para este mercado. Gran parte de ello fue el despliegue de VoLTE en Europa, Oriente Medio y África (EMEA) y Asia y el Pacífico.

Cuarenta y ocho redes comerciales VoLTE se han puesto en marcha a nivel mundial a partir del cierre del ejercicio 2015: 23 en Asia y Pacífico, 20 en EMEA y 5 en América del Norte. Los grandes fabricantes que participaron en estos desplieges, Ericsson, Alcatel-Lucent, Huawei, ZTE y BroadSoft registraron ganancias de ingresos anuales de más del 35 %.

Los datos anteriores confirman que IMS ha llegado para quedarse y ser referente mundial de una arquitectura VoIP.

4.2. Objetivos

El objetivo principal de este Proyecto Fin de Carrera es el estudio práctico de las funciones principales de un sistema IMS a través del diseño e implementación de un sistema funcional basado en herramientas de código abierto y virtualización.

Para llegar a este objetivo, no obstante, debemos iniciarnos primero en el mundo del protocolo que IMS utiliza para la gestión y establecimiento de sesiones: SIP.

Así, este proyecto tiene los siguientes objetivos:

1. Estudio teórico del protocolo SIP y de su arquitectura.
2. Estudio teórico y práctico del impacto de la interacción entre SIP y NAT.
3. Estudio práctico de diferentes escenarios desarrollados sobre virtualización (máquinas virtuales y virtualización ligera mediante kernel namespaces). Partiendo de un escenario simple con una llamada directa entre UAs, iremos incorporando elementos de la arquitectura SIP como servidores DNS, proxies SIP y servidores para NAT traversal (STUN, TURN, ICE, RTPProxy).
4. Estudio teórico y práctico de las medidas de seguridad que introduce SIP relativa a las comunicaciones (SRTP, SIP sobre TLS/DTLS, SIPS).
5. Estudio teórico y práctico de la arquitectura IMS haciendo énfasis en las funciones principales: P-CSCF, S-CSCF, I-CSCF y HSS. Además se han analizado todas las cabeceras específicas de SIP para IMS.
6. Implementación de un servidor de aplicaciones (AS) integrable en el escenario IMS que desarrolle la función de presencia basado en Kamailio.
7. Implementación de un servidor de aplicaciones (AS) integrable en el escenario IMS que desarrolle la función de buzón de voz basado en Asterisk.

4.3. Estructura de la memoria

Los dos grandes bloques de este proyecto, SIP e IMS, se han estructurado de forma similar. Una primera parte en la que se hace un estudio teórico y una segunda parte en la que se analiza y profundiza en los conceptos teóricos a través del desarrollo de prácticas.

La estructura es la siguiente:

- **SIP:** se centra en el estudio teórico de SIP y su interacción con NAT y DNS. Al final del capítulo se haya una sección con la guía de las prácticas desarrolladas y que analizan los conceptos vistos en teoría.
- **IMS:** se centra en el estudio teórico de IMS.
- **Implementación de un sistema IMS:** Recoge el proceso de implementación del escenario y de configuración de cada uno de los nodos. Al final de este capítulo se haya el guión de las prácticas desarrolladas.
- **Conclusiones:** Contiene las conclusiones del trabajo realizado y las líneas futuras.
- **Anexo - Kamailio:** Dada la fuerte presencia de este aplicativo, se veía la necesidad de incluir un capítulo propio. Se ha decidido añadirlo en los anexos dado que se trata de una de las herramientas utilizadas para el desarrollo de los escenarios.

Capítulo 5

SIP

SIP es un protocolo de señalización utilizado ampliamente para el establecimiento, gestión y finalización sesiones multimedia como voz, vídeo, mensajería instantánea o juegos en línea que nace de la combinación de dos protocolos:

- SIPv1: Protocolo diseñado para videoconferencias que incluye la ubicación de un usuario.
- SCIP: Protocolo para el establecimiento de sesiones multimedia basado en HTTP con identificación de usuario basada en su dirección de correo.

Definido por el IETF en el RFC3261[9], tiene sus raíces en la comunidad IP en lugar de en el sector de las comunicaciones por lo que presenta varia ventajas frente a otros protocolos:

- Se trata de un protocolo de propósito general no limitado sólo a telefonía IP.
- Señalización "simple" basada en texto.
- Permite incluir información adicional al propio estándar como información de presencia u otros protocolos para definición de sesiones multimedia (como SDP).
- Permite movilidad gracias al registro periódico centralizado que realizan los usuarios informando de su ubicación.
- Delega gran parte del soporte de los servicios en los usuarios, a diferencia de los protocolos de la telefonía tradicional como SS7.

Gracias a ellas, SIP se convierte en el protocolo de señalización de más amplia aceptación en VoIP reemplazando a H.323 (de la ITU-T) y en Noviembre de 2000 es seleccionado por el grupo 3GPP como protocolo de señalización para el IMS (IP Multimedia Subsystem).

Entre sus principales funciones encontramos:

- Ubicación: SIP determina ubicaciones de los usuarios por un proceso de registro permitiendo la existencia de múltiples ubicaciones registradas simultáneamente.
- Disponibilidad de los usuarios: Se trata de un método que permite determinar si existe o no un usuario y si está dispuesto a responder a una solicitud. Por ejemplo, un usuario puede tener varias ubicaciones registradas pero sólo aceptar comunicaciones entrantes en una de ellas. Algunos de los estados son disponible, ocupado o DND (Do not Disturb).
- Capacidades de usuario: Permite a un UA informar de las funcionalidades soportadas.
- Establecimiento de sesiones: Proporciona los medios para configurar y / o establecer la comunicación.
- Adminsitración de sesiones: Permite al UA finalizar una comunicación, transferirla o hacer modificaciones sobre los parámetros ya establecidos.

5.1. Arquitectura SIP

Existen dos tipos de entidades fundamentales:

1. **Agentes de Usuario (UA)**:

   Entidades finales capaces de gestionar señalización SIP y sesiones y flujos multimedia (mediante el uso de otros protocolos como SDP y RTP).

2. **Servidores SIP**: Servidor Registrar, Proxy SIP y Servidor Redirect

   Equipos intermedios que gestionan el intercambio de mensajes SIP. Entre otras funciones ponen en contacto UAs pero no intervienen en flujos multimedia.

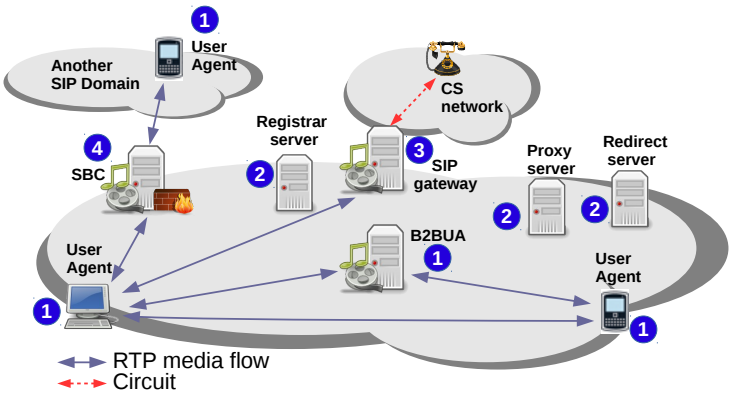


Figura 5.1: Arquitectura SIP.

La figura 5.1 muestra los elementos más comunes que podemos encontrar en la comunicación SIP y que a continuación veremos más en detalle.

Agentes de Usuario (UA)

Como hemos mencionado, el User Agent (UA) es una entidad capaz de gestionar señalización SIP y sesiones y flujos multimedia y que está compuesto a su vez por:

- **UAC - User agent client**: Entidad lógica que genera y envía peticiones.
- **UAS - User Agent Server**: Entidad lógica que genera respuestas.

Durante una sesión, un UA por lo general opera tanto como UAC y UAS y, a diferencia de los servidores SIP, también gestiona flujos multimedia por lo que debe ser capaz de describir dichos flujos. Es decir, el UA debe ser capaz de gestionar un protocolo de descripción de sesiones multimedia y un protocolo para su transporte. Típicamente SIP se combina con Session Description Protocol (SDP) para la gestión de las sesiones multimedia y Real-time Transport Protocol (RTP) para su transporte.

Un UA no tiene porque situarse en los extremos de la comunicación si no que puede tratarse de un elemento intermediario. Decimos, sin embargo, que se trata de entidades finalistas dado que si que lo son desde el punto de vista de sesión y flujo multimedia.

Un ejemplo de un UA como elemento intermedio es el denominado **Back to Back User Agent (B2BUA)** que no es más que una concatenación de un UAS y UAC capaz por una parte de finalizar una pata de una llamada y por la otra generar otra pata de la llamada reenviando entre ellas el tráfico de sesión y flujo multimedia. En la figura 5.2 podemos ver un ejemplo del flujo de una sesión SIP que atraviesa esta entidad.

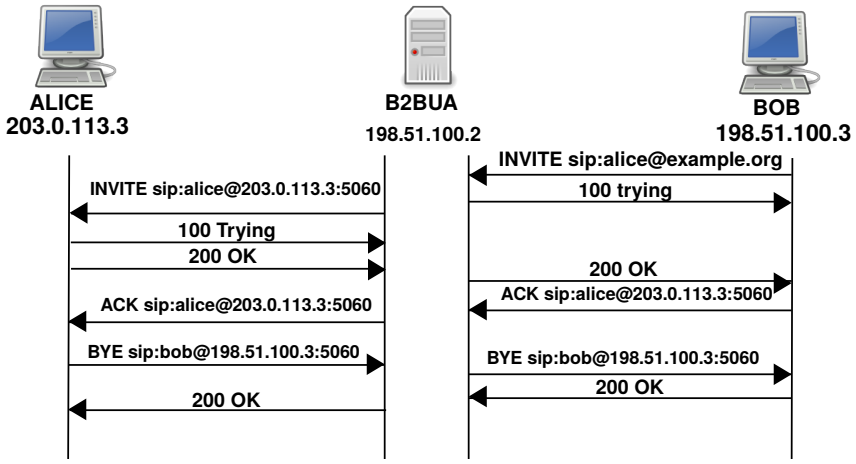


Figura 5.2: Flujo SIP a través de un B2BUA.

Dos ejemplos clásicos de implementación de un B2BUA son:

1. **Gateways**:
   - Permiten el intercambio de llamadas entre redes de circuitos y redes SIP.
   - Gestionan traducciones de SIP a Señalización CS (SS7).
   - Gestionan transcodificación de flujos multimedia.
2. **Session Border Controller (SBC)**:
   - El SBC es un dispositivo ”frontera” (Border) habitualmente situado, como muestra la figura 5.3, entre dominios administrativos o entre la red de acceso y la red troncal del proveedor SIP.
   - Realiza funciones de “firewall“ ejerciendo control no sólo sobre la señalización SIP si no también sobre los flujos multimedia. Puede, por ejemplo, atendiendo a politicas de tarificación acabar una llamada o prohibirla.
   - A menudo proporciona mecanismos de medición, control de acceso, y, en el caso de gestión flujos, mecanismos de transcodificación.
   - También son una posible solución al problema del NAT (Network Address Translation).

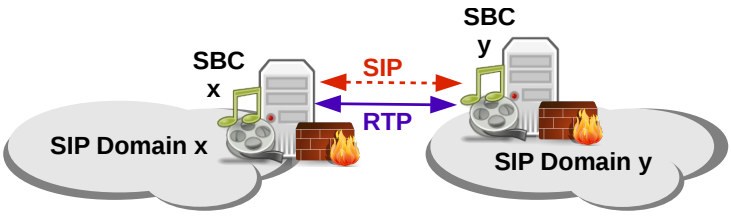


Figura 5.3: SBC como frontera de dominios administrativos.

Servidor Registrar

El proceso de registro permite dotar de movilidad a los usuarios SIP aunque cabe destacar que dicha movilidad no aplica en llamadas establecidas (call continuity).

Cada usuario tiene una **Address-of-Record (AOR)** que no es más que una URI SIP (o SIPS) que apunta a un dominio donde hay un servicio de localización. Es decir, la AOR se puede entender como la ”dirección pública“ del usuario.

El servicio de localización permite mapear la URI del AOR a otra URI donde el usuario puede estar disponible. Típicamente, el servicio de localización se provee de datos mediante el proceso de registro entre el UA y un servidor registrar.

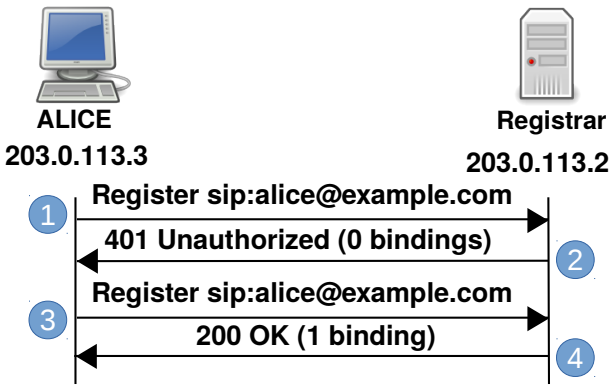


Figura 5.4: Proceso de registro.

La figura 5.4 muestra el proceso de registro de un UA:

1. El UA envía una petición con el método REGISTER:
   - La petición se encamina al servidor de registro indicado en la **request URI**.
   - Informa de qué dirección de contacto (header **`Contact`**) debe asociarse al AOR (header **`To`**).

```
REGISTER sip:203.0.113.2 SIP/2.0
Via: SIP/2.0/UDP 203.0.113.3:5060;rport;branch=
z9hG4bKPjPUuQdSLFvSs62wtvszZq5QtM3yHhaYNK
Route: <sip:203.0.113.2;lr>
Max-Forwards: 70
From: <sip:alice@example.org>;tag=OeY3VoysTt6wcFhDEhXt9ghk7tfJah4r
To: <sip:alice@example.org>
Call-ID: b472q0.De5q0f4eTP2JXo2Jcku.1hNDp
CSeq: 26552 REGISTER
User-Agent: PJSUA v2.3 Linux-3.3.8/i686/glibc-2.11
Contact: <sip:alice@203.0.113.3:5060;ob>
Expires: 0
Content-Length:  0
```

2. Si el registrar ha sido configurado con **autenticación** enviará una respuesta "401 Unauthorized" para indicar que es necesaria y especificando el mecanismo para llevarla a cabo:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 203.0.113.3:5060;rport=5060; branch=
z9hG4bKPjPUuQdSLFvSs62wtvszZq5QtM3yHhaYNK;received=203.0.113.3
From: <sip:alice@example.org>;tag=OeY3VoysTt6wcFhDEhXt9ghk7tfJah4r
To: <sip:alice@example.org>;tag=b27e1a1d33761e85846fc98f5f3a7e58.403e
Call-ID: b472q0.De5q0f4eTP2JXo2Jcku.1hNDp
CSeq: 26552 REGISTER
WWW-Authenticate: Digest realm="example.org",
nonce="VhJxxlYScJrjisKts5M7fWpHfkKFxoqS"
Server: kamailio (4.2.6 (i386/linux))
Content-Length: 0
```

   - En este ejemplo se utiliza autenticación de tipo WWW (header **`WWW-Authenticate`**) en la que el UA y el servidor registrar comparten una contraseña.
   - El registrar envía un reto (nonce) para ser relacionado con el password y con el que el UA realizará las siguientes operaciones para generar la respuesta al reto de autenticación:

$$HA1 = MD5(username : realm : password)$$
$$HA2 = MD5(method : digestOfRequestURI)$$
$$response = MD5(HA1 : nonce : HA2)$$

3. El UA vuelve a enviar la petición REGISTER, esta vez con los datos necesarios para la autenticación:

```
REGISTER sip:203.0.113.2 SIP/2.0
Via: SIP/2.0/UDP 203.0.113.3:5060;rport;branch=
z9hG4bKPjqwLrEeKmVZc0G1UXtjabLcPp9Ff4Ib06
Route: <sip:203.0.113.2;lr>
Max-Forwards: 70
From: <sip:alice@example.org>;tag=OeY3VoysTt6wcFhDEhXt9ghk7tfJah4r
To: <sip:alice@example.org>
Call-ID: b472q0.De5q0f4eTP2JXo2Jcku.1hNDp
CSeq: 26553 REGISTER
User-Agent: PJSUA v2.3 Linux-3.3.8/i686/glibc-2.11
Contact: <sip:alice@203.0.113.3:5060;ob>
Expires: 0
Authorization: Digest username="alice", realm="example.org",
nonce="VhJxxlYScJrjisKts5M7fWpHfkKFxoqS", uri="sip:203.0.113.2",
response="e3dcbb062fe3e9b53357de3aa741864e"
Content-Length:  0
```

4. Si el proceso de validación es correcto, el registrar realiza una asociación (binding) y envía una respuesta 200 OK:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 203.0.113.3:5060;rport=5060;branch=
z9hG4bKPjqwLrEeKmVZc0G1UXtjabLcPp9Ff4Ib06;
received=203.0.113.3
From: <sip:alice@example.org>;tag=OeY3VoysTt6wcFhDEhXt9ghk7tfJah4r
To: <sip:alice@example.org>;tag=b27e1a1d33761e85846fc98f5f3a7e58.021a
Call-ID: b472q0.De5q0f4eTP2JXo2Jcku.1hNDp
CSeq: 26553 REGISTER
Server: kamailio (4.2.6 (i386/linux))
Content-Length: 0
```

En general el registro puede tener un período de vigencia limitado en cuyo caso el binding caduca si no es renovado. El cliente propone un tiempo de vigencia indicado en segundos en el header **`Expires`** pero es el registrar el que finalmente finalmente lo asigna.

En el ejemplo anterior, el valor ”0“ del header **`Expires`** indica un tiempo ilimitado. El servidor registrar lo ha aceptado dado que no ha enviado ese campo en la respuesta corrigiendo así el valor.

Un binding también puede ser finalizado mediante un proceso de desregistro.

Proxy SIP

Un servidor proxy es una entidad intermediaria que encamina peticiones en nombre de clientes (UAs). Sus principales características son:

- Garantiza que las peticiones se envían a otra entidad más cercana al UA destino.
- Interpreta y, si es necesario, reescribe partes específicas de las peticiones antes de reenviarlas.
- En general, no inicia peticiones SIP propias, sólo las reenvía aunque existen excepciones como la generación de peticiones ACK y CANCEL en determinados casos (e.j. forking proxy).
- Puede responder peticiones de UAs.
- No tiene capacidades multimedia.
- Puede aplicar políticas a nivel SIP. Por ejemplo, antes de reenviar una petición, puede asegurarse que al UA correspondiente se le permite establecer la sesión.

Hay diferentes cualidades según las que se clasifican los proxy sip.

Según la información que guardan se pueden clasificar en:

- **Stateless:** Procesan cada mensaje sin estado o contexto. Es decir, un proxy stateless no podría correlar una petición con su correspondiente respuesta ya que realiza un simple reenvío (forward) de mensajes sin guardar ningún tipo de información.

  Al no mantener estado son altamente escalables pero su funcionalidad es limitada. Básicamente se utilizan para realizar repartos de carga ya que no pueden ofrecer funcionalidades avanzadas como “forward on busy”, buzón de voz o contabilidad de llamadas (call accounting) entre otras.

- **Transaction stateful:** Mantienen en memoria información relacionada con cada una de las transacciones que procesan hasta que finalizan lo que les permite reconocer respuestas, fallos o retransmisiones y ejecutar rutinas en base a ello.

  No mantienen, sin embargo, información sobre diálogos por lo que no pueden, por ejemplo, calcular la duración de una llamada en tiempo real (call accounting).

  La figura 5.5 muestra el comportamiento de un proxy transaction stateful.

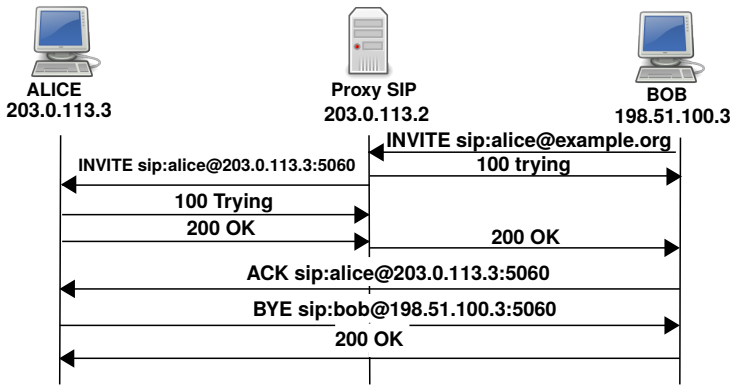


Figura 5.5: Proxy transaction stateful.

- **Call stateful:** Mantienen información de todo el diálogo SIP (llamada) lo que permite calcular duraciones y terminar llamadas en tiempo real (call accounting).

  Es el menos escalable pero el que más funcionalidad ofrece. No son B2BUA pero, al igual que éstos, necesitan estar presentes en todo momento en el camino de señalización de la llamada. Para ello hace uso de la cabecera `Record-Route`.

  La figura 5.6 muestra el comportamiento de un proxy call stateful.

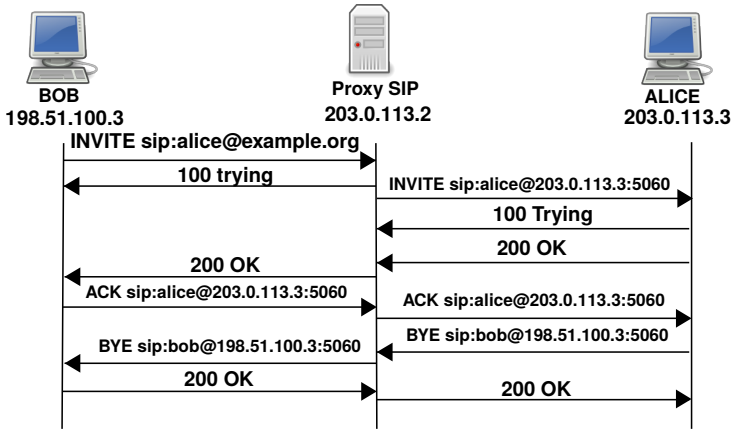


Figura 5.6: Proxy call stateful.

Según si gestionan peticiones salientes o entrantes a un dominio administrativo se pueden clasificar en:

- **Outbound:** Un outbound proxy ayuda a los UAs a encaminar sus peticiones SIP salientes. De hecho, los UAs generalmente están configurados para encaminar todas sus peticiones SIP a través de un outbound proxy.

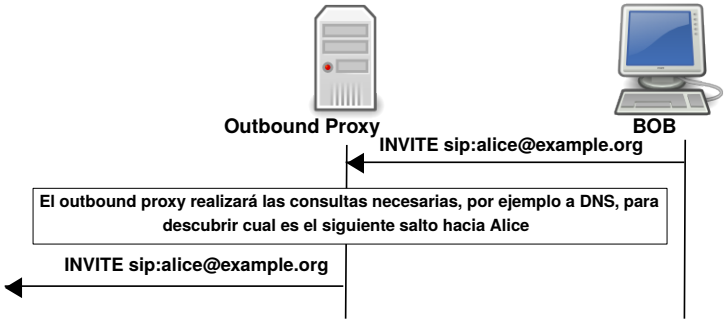


Figura 5.7: Outbound proxy.

- **Inbound:** Procesan peticiones SIP de entrada a un dominio. Como muestra la figura 5.8, consultan el servicio de localización, determinan la URI de contacto del UA destino y reenvían la solicitud a dicha localización.

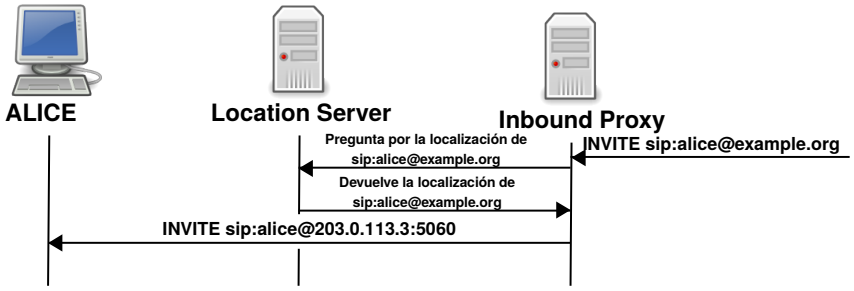


Figura 5.8: Inbound proxy.

Dos escenarios frecuentes en los que intervienen servidores proxy SIP son:

- **Trapecio SIP**: Es aquel en el que las peticiones, en su camino desde el originador al destinatario, atraviesan un outbound y un inbound proxy. Este escenario, mostrado en la figura 5.9, se conoce comúnmente como ”trapecio SIP“ (SIP Trapezoid) por la forma trapezoidal que dibujan el camino SIP y el del flujo multimedia.

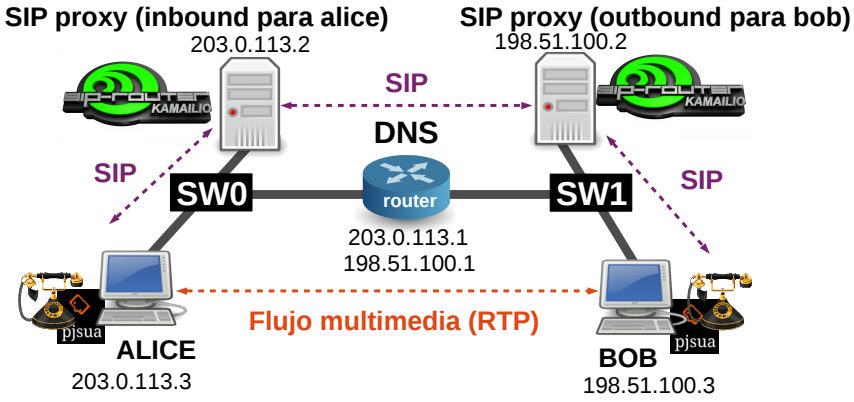


Figura 5.9: Trapecio SIP.

- **Forking Proxy**: Posibilita llamar a un usuario que está registrado en más de una ubicación ya que, tal y como muestra la figura 5.10, al recibir una petición, genera tantas como ubicaciones hayan registradas. Cuando

una ubicación contesta, el proxy cancela las demás enviando peticiones de tipo CANCEL. Cada UA que recibe un CANCEL envía una respuesta con código 487 y, finalmente, cada transacción se cierra con un ACK que tiene el mismo `branch` que el CANCEL original. Es por ello que un forking proxy es, como mínimo, transaction stateful y actúa como inbound proxy.

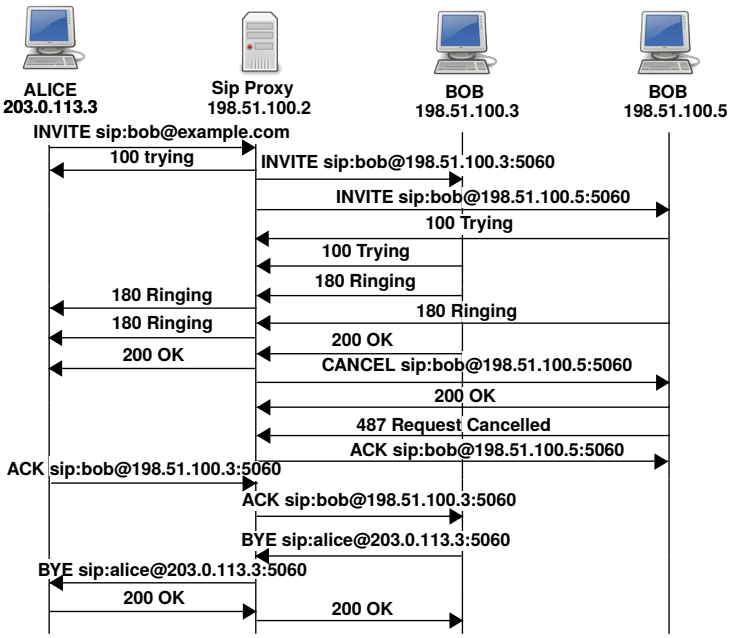


Figura 5.10: Forking Proxy.

Servidor Redirect

Suministran ubicaciones alternativas en las que puede localizarse al usuario.

Al contrario que un proxy SIP, no inicia ninguna acción para determinar las localizaciones activas del usuario si no que se limita a devolver al UA una lista de posibles ubicaciones en las que se podría localizar al usuario y es el UA el que debe generar tantas peticiones como localizaciones haya recibido. La figura 5.11 muestra un ejemplo de este comportamiento.

No requieren, por tanto, ser call stateful por lo que tienen una alta capadidad de procesado.

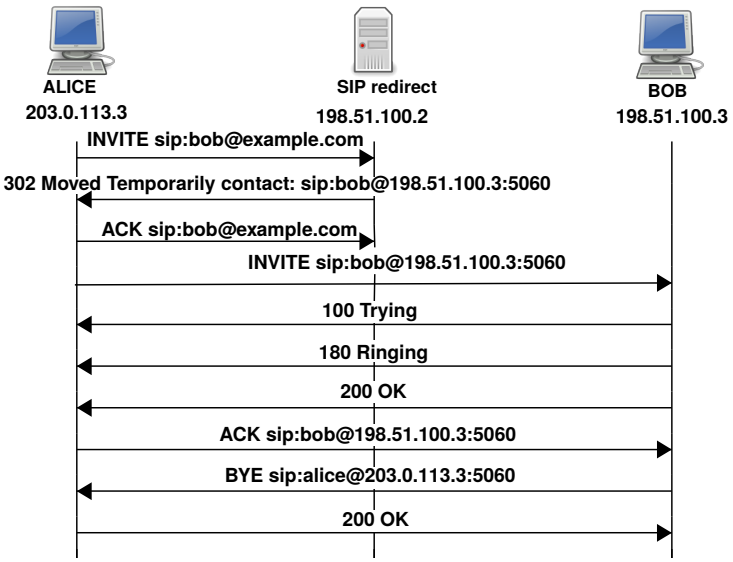


Figura 5.11: Servidor redirect.

Recordar que en este caso el primer ACK es parte de la transacción INVITE (tiene el mismo `branch`) y por ello el destino es el servidor redirect.

5.2. Fundamentos SIP

Sip sigue el modelo cliente - servidor basado en el intercambio de peticiones, también denominados métodos, y respuestas.

5.2.1. Métodos

Las peticiones SIP también se denoniman métodos (methods). El protocolo base de SIP (RFC3261[9]) define seis métodos:

1. *INVITE*: Inicia el establecimiento de una sesión de comunicación entre UAs.
2. *ACK*: Confirmación de recepción de una respuesta SIP.
3. *CANCEL*: Termina una solicitud pendiente. Es decir, una sesión que no ha sido establecida previamente.
4. *BYE*: Finaliza una sesión establecida.
5. *REGISTER*: Utilizado por los UAs para registrarse en un servidor SIP "registrar".
6. *OPTIONS*: Solicita información a otro UA sin establecer una sesión. Esta información puede ser relativa a métodos soportados, tipos de contenidos, extensiones, codecs, etc. La información es proporcionada dentro de una respuesta 200 OK.

En RFCs adicionales se extienden con nuevos métodos para ofrecer funcionalidades adicionales. Los principales métodos extendidos son:

- *SUBSCRIBE*: Petición de suscripción para recibir notificaciones referentes a los eventos de un usuario SIP (presencia).
- *NOTIFY*: Notifica eventos a un suscriptor.
- *PUBLISH*: Publica un evento en un servidor de presencia. El servidor es el que generará tantos NOTIFY como suscriptores existan.
- *REFER*: Indica que se desea ser notificado del resultado de otra petición. Típicamente se usa en transferencias de llamadas.
- *INFO*: Permite enviar información de control a nivel de aplicación en una sesión establecida. Su uso más extendido es el envío de tonos DTMF por el camino de señalización SIP. Los clientes deciden si envían los tonos DTMF por medio de RTP (RFC2833[10]) o SIP.
- *UPDATE*: Modifica el estado de una sesión multimedia y, a diferencia de un re-INVITE, se puede utilizar antes de que se haya completado el 3-way-handshake (ej. anuncios).
- *MESSAGE*: Transporta mensajes instantáneos a través de SIP.
- *PRACK*: SIP no incluye ningún mecanismo para entregar respuestas provisionales de forma fiable y hay casos donde es importante asegurar la entrega de una respuesta provisional. Por ejemplo, en el interfuncionamiento SIP/PSTN es crucial que las respuestas 180 (ringing) y 183 (Session in Progress) no se pierdan. Este método garantiza una entrega fiable y ordenada de respuestas provisionales en SIP.

5.2.2. Respuestas

Las respuestas SIP son muy similares a las respuestas de HTTP. Se componen de un código numérico y una descripción y se dividen en seis categorías en función del primer dígito del código de respuesta:

- **Respuestas 1XX:** Son respuestas provisionales e informativas previas al establecimiento de una sesión. Algunos códigos de respuesta 1XX habituales son: 100 (Trying), 180 (Ringing), 181 (Call is Being Forwarded), 182 (Queued y, 183 (Session in Progress).

- **Respuestas 2XX:** Respuestas afirmativas que indican la aceptación del método de la petición. Códigos de respuesta 2XX habituales: 200 (OK), 202 (Accepted) y 204 (No Notification).
- **Respuestas 3XX:** Indican que una acción adicional es necesaria para completar la petición. Por ejemplo una redirección. Códigos de respuesta 3XX habituales: 300 (Multiple Choices), 301 (Moved Permanently), 302 (Moved Temporarily), 305 (Use Proxy) y 380 (Alternative Service).
- **Respuestas 4XX:** Indican que no se puede establecer la sesión por algún tipo de error en el lado del cliente. Códigos de respuesta 4XX habituales: 400 (Bad Request), 401 (Unauthorized), 402 (Payment Required), 403 (Forbidden), 404 (Not Found), 405 (Method Not Allowed), 406 (Not Acceptable), 407 (Proxy Authentication Required), 408 (Request Timeout), 486 (Busy Here), 487 (Request Terminated)..
- **Respuestas 5XX:** Indican algún tipo de error en un servidor SIP como por ejemplo que no hay canales de datos disponibles, sobrecarga en la red o numeración no alcanzable. Códigos de respuesta 5XX habituales: 500 (Server Internal Error), 501 (Not Implemented), 503 (Service Unavailable), y 504 (Server Time-out).
- **Respuestas 6XX:** Indican algún tipo de fallo global. Estos códigos se usan cuando la solicitud no puede ser procesada por ningún servidor. Códigos de respuesta 6XX habituales: 600 (Busy Everywhere), 603 (Decline), 604 (Does Not Exist Anywhere), y 606 (Not Acceptable).

5.2.3. Mensajes SIP

La mayor parte de la sintaxis de los mensajes y campos de las cabeceras SIP son muy similares a HTTP. Están basados en texto y utilizan codificación UTF-8 (conjunto de caracteres ISO 10646).

El formato de los mensajes SIP, mostrado en la figura 5.12, se define en el RFC2822[6] y están compuestos por:

- Línea inicial (Start-Line).
- Uno o más campos de cabecera (Message Headers)
- Línea vacía (CR+LF) que indica el final de las cabeceras.
- Opcionalmente un cuerpo del mensaje (Message Body)

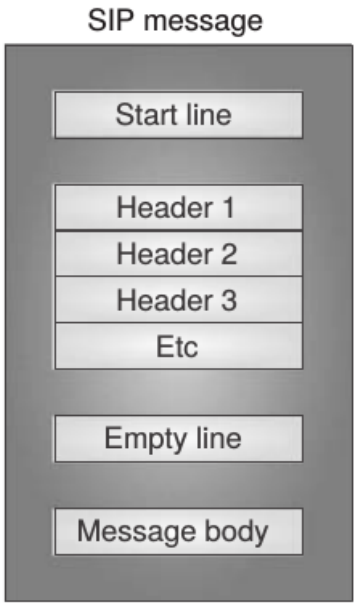


Figura 5.12: Estructura de los mensajes SIP.

La primera línea de un mensaje define que tipo de mensaje es.

En las peticiones la primera linea se llama **request line** y contiene el nombre del método.

```
INVITE sip:alice@203.0.113.2 SIP/2.0
Via: SIP/2.0/TCP
198.51.100.3:54274;rport;branch=z9hG4bKPjImxzxCydmO63BYTMt3Qt6yEm1HL5IBZw;alias
Max-Forwards: 70
From: sip:bob@example.com;tag=vJtmQaD5mHLCIBlAeOdf55d.Mm.bOmXV
To: sip:alice@203.0.113.2
Contact: <sip:bob@198.51.100.3:5060;ob>
....
```

En las respuestas la primera linea del mensaje siempre empieza por la cadena "SIP/", a continuación la versión y finalmente el código de estado en relación al método solicitado.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP
198.51.100.3:54274;rport=54274;received=198.51.100.3;
branch=z9hG4bKPjImxzxCydmO63BYTMt3Qt6yEm1HL5IBZw;alias
Record-Route: <sip:203.0.113.2;lr>
Record-Route: <sip:198.51.100.2;lr;r2=on>
Record-Route: <sip:198.51.100.2;transport=tcp;lr;r2=on>
...
```

Request-line

Contiene el nombre del método (INVITE, BYE, etc.), un identificador del destinatario de la solicitud denominado Request-URI, la versión del protocolo SIP y un retorno de carro y salto de línea (caracteres CR+LF):

```
INVITE sip:bob@example.com SIP/2.0
```

SIP usa URIs (Uniform Resource Indicators) para encaminar peticiones y aparecen referenciadas en diversas cabeceras de los mensajes SIP. El formato de las URIs para SIP es el siguiente:

```
sip:user:password@host:port;uri-parameters?headers

Por ejemplo:
sip:alice@example.org:1234;user=phone?priority=urgent
```

- **user:password**: Identifica a un usuario y su contraseña en el dominio en el que se hace la solicitud. El password es opcional.
- **host**: Identifica al host que suministra un recurso particular. Puede ser un nombre de dominio, una dirección IPv4 o una dirección IPv6.
- **port**: Identifica el puerto de servicio al que se envía la solicitud.
- **uri-parameters**: Parámetros adicionales separados por puntos y comas. Los más habituales son `transport`, `maddr`, `ttl`, `user`, `method` y `lr`. El parámetro `transport` especifica el protocolo de transporte (udp,tcp o sctp) mientras que el parámetro `lr` indica un tipo de encaminamiento no estricto que en el apartado Routing en SIP veremos en detalle.
- **headers**: Se especifican detrás del signo "?" y se codifican como parejas de valores `hname=hvalue` separados por ampersands (&). Sirven en el caso en que se utilice la URI para construir nuevas peticiones. Es decir, los headers presentes en la SIP URI deben incluirse en nuevas peticiones. Un caso particular es el que que hname es "body" e indica que el valor es el propio cuerpo (message-body) de la petición SIP.

Cabeceras

A continuación de la "Request-line" aparece la cabecera SIP. Está compuesta por uno o más campos, también denominados **SIP fields**, que proporcionan los atributos del mensaje SIP y que, de nuevo, son similares a los campos de las cabeceras HTTP.

Todos ellos siguen el formato:

```
Header: Value1;Value2;...
```

Los campos básicos que podemos encontrar en la cabecera SIP son:

- **`From`** y **`To`**: Campos obligatorios. Identifican respectivamente al emisor y al receptor de una petición. Esto es, cuando el receptor genera una repuesta, **`From`** y **`To`** no se invierten.
- **`Via`**: Campo obligatorio. Como muestra el ejemplo de la figura 5.13, son los primeros que aparecen en los mensajes SIP tras la start line. Sirven para identificar transacciones, encaminar respuestas e identificar y evitar bucles. Cada campo **`Via`** contiene:
  - La versión de SIP y el protocolo de transporte.
  - La dirección IP y puerto por donde ha pasado el mensaje SIP.
  - Un string aleatorio en el parámetro `branch` que identifica la transacción y permite correlar las respuestas a una petición.

  Existen dos parámetros especiales, `received` y `rport`, que se utilizan cuando un servidor recibe una solicitud y la dirección IP de origen, en el caso de `received`, o el puerto, para `rport`, del paquete recibido no se corresponden con la dirección IP o con el puerto indicados en la cabecera **`Via`**. Para poder encaminar correctamente la respuesta, el servidor agrega, según sean necesarios, el parámetro `received` con la dirección IP y / o el campo `rport` con el puerto observados.

  Ejemplo de cabeceras **`Via`**:

```
Via: SIP/2.0/UDP 200.20.20.2;branch=z9hG4bKb06c.0597.0
Via: SIP/2.0/UDP 10.20.20.4:5060;received=10.200.15.253;branch=z9hG523;received=198.51.100.132;rport=56789
```

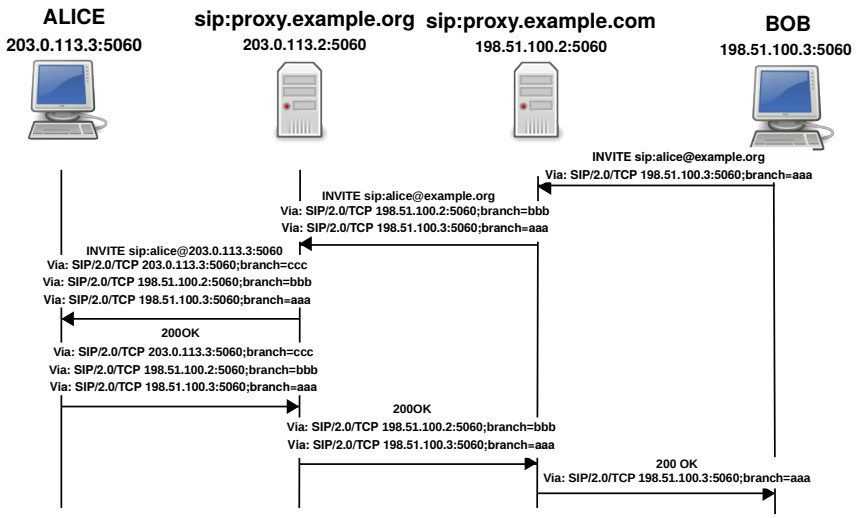


Figura 5.13: Cabecera Via.

- **`Contact`**: Contiene una SIP URI que indica a un UA remoto dónde enviar futuras peticiones SIP dentro de un mismo diálogo. Esta es la dirección SIP por defecto y permite a los proxies **no estar en el camino en las subsiguientes peticiones** una vez ponen en contacto a los UA. La figura 5.14 muestra un ejemplo de este comportamiento.

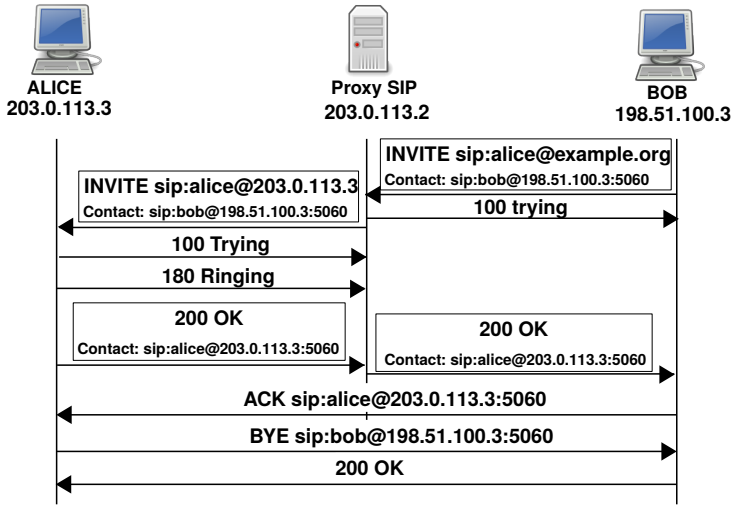


Figura 5.14: Cabecera Contact.

- **`Record-Route`**: Modifican el comportamiento anterior. Un proxy puede querer permanecer en la señalización de forma que futuras peticiones del diálogo sean enviadas a través de él. Para ello utiliza la header **`Record-Route`** e indica en él su URI SIP.

  El UA que recibe la petición copia en la respuesta los headers **`Record-Route`** recibidos para que éstos sean recibidos también por el emisor y así indicar que futuras peticiones deben enviarse a través del camino dictado por la pila de encabezados.

  Es decir, las cabeceras **`Record-Route`** no son necesarias para determinar la trayectoria de las respuestas si no de futuras peticiones y, a diferencia de las cabeceras **`Via`**, no serán eliminadas cuando la respuesta se dirija al UA origen.

  Cuando el UA origen reciba la respuesta de la primera transacción, ambos extremos tendrán la misma pila de cabeceras **`Record-Route`** (lista de puntos intermedios).

- Cabecera **`Route`**: En general, una pila de cabeceras **`Route`** fuerza a enrutar las peticiones SIP a través de la lista de proxies proporcionada.

  Los mecanismos **`Record-Route`** y **`Route`** funcionan de la siguiente forma:

  - Cuando un UA ha de generar una nueva petición incluye la pila de rutas aprendidas cambiando **`Record-Route`** a **`Route`**.
    - El UA destino, en términos de diálogo, incluye la pila de rutas directamente.
    - El UA origen, en términos de diálogo, incluye la pila de rutas en orden inverso.
  - Cada punto intermedio elimina la cabecera **`Route`** que le hace referencia.

  La figura 5.15 muestra un ejemplo gráfico de dicho comportamiento:

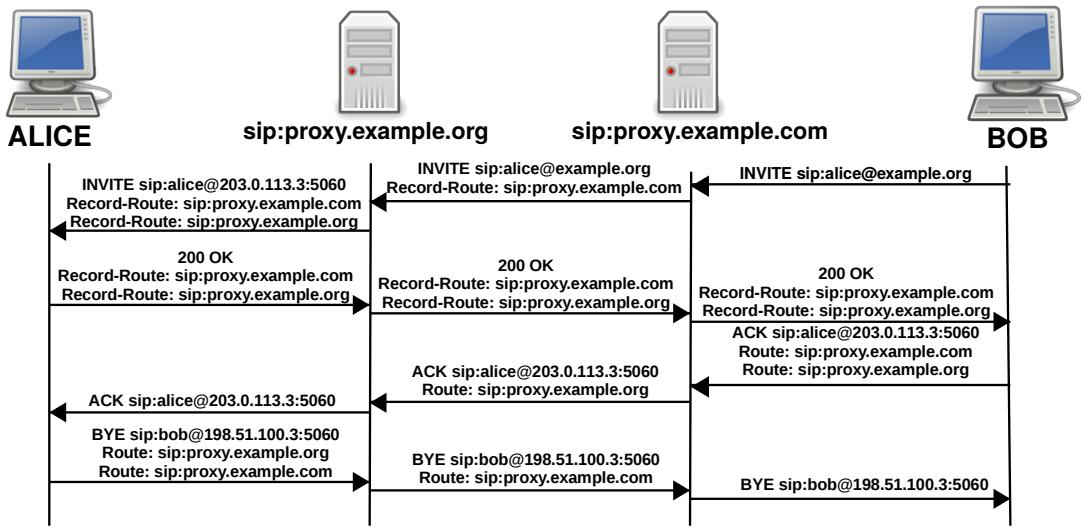


Figura 5.15: Cabeceras Record-route y Route.

La ruta también se puede aprender de otras formas, por ejemplo, preconfigurada o durante el registro del UE mediante las cabeceras **`Path`** y **`Service-Route`** usadas en la arquitectura IMS.

- **`Cseq`**: Campo obligatorio. Se trata de un número de secuencia del mensaje que se utiliza para ordenar las peticiones extremo a extremo dentro de un mismo diálogo. Esto es, un UA debe incrementar en 1 cada nueva petición que envía. De esta forma, el UA receptor puede determinar si una petición ha llegado fuera de orden.

  Como excepción, las peticiones ACK y CANCEL no causan un incremento en el número de secuencia dado que estas peticiones no siempre son extremo a extremo. Un ejemplo de ello son los forking proxies que al recibir una petición pueden generar múltiples del mismo tipo pero con destino diferente. El primer destino del que se reciba una respuesta positiva será con el que se establezca el diálogo. Sin embargo, la petición sólo será cancelada en caso de que se reciba una respuesta negativa desde todos y cada uno de los destinos.

  Las peticiones ACK tienen el mismo número en **`CSeq`** que el método que reconocen mientras que las peticiones CANCEL tienen el mismo número en **`CSeq`** que la transacción que cancelan.

  Para poder distinguir una respuesta a un CANCEL de una respuesta a un INVITE, por ejemplo, la cabecera **`CSeq`** incluye, a continuación del número de secuencia, el nombre del método al que hace referencia.

- **`MaxForwards`**: Campo obligatorio. Similar al TTL en IP, contiene el número máximo de saltos SIP permitidos y permite minimizar la carga provocada por bucles.

- **`Call-ID`**: Campo obligatorio. Identificador único que identifica los mensajes SIP que pertenecen a un mismo diálogo o registro. Dado que elementos intermedios como los B2BUA modifican este identificador, SIP incorpora mecanismos adicionales de identificación de sesión (**`Session-ID`**).

- **`Content-Type`** y **`Content-Length`**: Se trata de campos opcionales que contienen la descripción y tamaño del cuerpo del mensaje cuando éste está presente. Por ejemplo:

```
Content-Type: application/sdp
Content-Length: 151
```

- Campo **`User-Agent`**: Contiene una descripción del tipo de terminal que origina el mensaje. Por ejemplo del fabricante y versión del softphone, ATA o teléfono IP.

```
User-Agent: PJSUA v2.3 Linux-3.3.8/i686/glibc-2.11
```

- **`Service-Route`**: Se suele utilizar en una respuesta 2xx a una petición REGISTER e indica al UA el primer salto (ruta) que debe emplear en futuras peticiones. Sólo es válida para la duración del registro y debe refrescarse cuando el registro se actualiza. Un ejemplo es:

```
Service-Route: <sip:proxy3.example.com;lr>
```

- **`Path`**: Es una cabecera opcional que puede aparecer en la petición REGISTER que llega a un servidor registrar. Mediante esta cabecera, un proxy SIP en el camino del registro, indica que desea estar en los futuros caminos SIP que sigan las peticiones al UA que se está registrando (p.e. INVITEs de llamadas entrantes). Esta información es utilizada por los proxies posteriores en el camino del registro para saber que deben utilizar el proxy que introduce la cabecera **`Path`** para enviar peticiones al UA correspondiente. Puede interpretarse como un mecanismo similar al **`Record-Route`** pero que sucede en el registro, es decir, previamente a las llamadas. Se utiliza extensivamente en la red IMS, ejemplo:

```
Path: <sip:proxy2.example.com;lr>
```

Message Body

El cuerpo de un mensaje SIP es opcional. Si existe, por lo general se trata de un mensaje del protocolo SDP que describe la comunicación multimedia:

```
 Ejemplo:
v=0
o=- 3653038228 3653038228 IN IP4 198.51.100.3
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 4000 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 198.51.100.3
b=TIAS:64000
a=rtcp:4001 IN IP4 198.51.100.3
a=sendrecv
a=rtpmap:98 speex/16000
a=rtpmap:8 PCMA/8000
a=fmtp:104 mode=30
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-16
```

5.2.4. Transacciones, diálogos y sesión

Se definen dos tipos diferentes de "conversaciones" entre elementos SIP: las transacciones y los diálogos.

Un **diálogo** se entiende como todos los mensajes relacionados con una "llamada" extremo a extremo mientras que una **transacción** es un intercambio petición/respuesta entre dos entidades adyacentes a nivel SIP. Una transacción, como muestra la figura 5.16, se inicia con una petición y termina, en general, con una respuesta final aunque puede contener cero o más respuestas provisionales. Como excepción, existe un método que puede no tener respuesta final. Es el caso del método ACK.

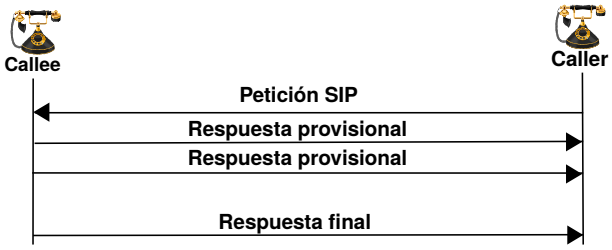


Figura 5.16: Transacción SIP.

Es decir, las transacciones son en realidad **partes de un diálogo** y sólo tienen significado entre dos dispositivos adyacentes a nivel SIP dado que las peticiones que atraviesan diferentes dispositivos SIP generan nuevas transacciones (valores `branch` diferentes).

La figura 5.17 muestra un ejemplo de la señalización de un único diálogo compuesto por cuatro transacciones.

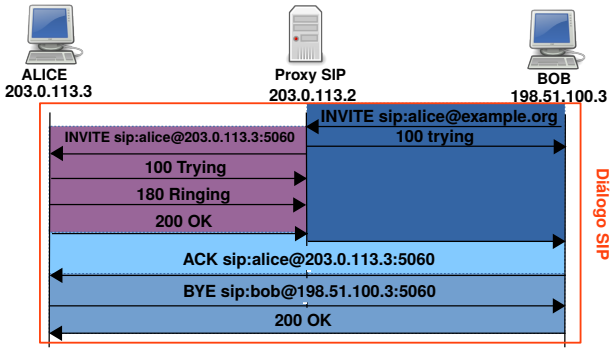


Figura 5.17: Ejemplos de transacciones.

Una transacción se identifica unívocamente por el campo `branch` del header **`Via`** mientras que un diálogo utiliza los campos `tag` de las cabeceras **`From`** y **`To`** y la cabecera **`Call-ID`** para tal propósito.

Cuando se inicia un diálogo, el emisor genera aleatoriamente un valor para el campo `tag` de la cabecera **`From`** y otro valor para la cabecera **`Call-ID`**.

```
INVITE sip:alice@203.0.113.2 SIP/2.0
...
From: sip:bob@example.com;tag=vJtmQaD5mHLCIBlAeOdf55d.Mm.bOmXV
To: sip:alice@203.0.113.2
Call-ID: xB4IsFv1I4fX296-c2kio3UkgYQTX3Uo
```

Al llegar al receptor, éste genera aleatoriamente otro valor para el campo `tag` de la cabecera **`To`** e incluye también los parámetros recibidos en la respuesta.

```
SIP/2.0 200 OK
...
From: <sip:bob@example.com>;tag=vJtmQaD5mHLCIBlAeOdf55d.Mm.bOmXV
To: <sip:alice@203.0.113.2>;tag=wyZIvK16DO8A04LtHQVXVZt3ahy46WK5
Call-ID: xB4IsFv1I4fX296-c2kio3UkgYQTX3Uo
....
```

Finalmente, una **sesión** es el flujo de media entre UAs. Notar que el establecimiento de una sesión utiliza un *3-way handshake* en el que intervienen dos transacciones: INVITE y ACK.

Transacciones comunes en SIP

- **Transacción INVITE**:
  - Se inicia con la petición INVITE.
  - Termina con una respuesta de código 200 OK.
  - Admite respuestas intermedias como 100 Trying y 180 Ringing.
- **Transacción BYE**:
  - Para terminar una sesión SIP un usuario libera la llamada enviando una petición BYE.
  - El otro extremo responde con una respuesta 200 OK.
- **Transacción ACK:**
  - En general, el ACK tiene su propia transacción y no requiere respuesta.
  - Sin embargo, cuando una respuesta final a una transacción previa no tiene un código de estado 2xx, entonces el ACK es parte de la transacción.
- **Transacción re-INVITE:**
  - Una petición INVITE también puede ser enviada dentro de un diálogo ya establecido. Se le demonina entonces re-INVITE.

- Una petición re-INVITE se usa típicamente para modificar los parámetros de una sesión en curso. Por ejemplo, en medio de una sesión, un usuario puede decidir agregar un componente de vídeo. En ese caso, el usuario debe enviar un nuevo INVITE (re-INVITE) dentro del mismo diálogo en el que estableció la sesión salvo que ahora el SDP incluye el componente de vídeo adicional.
- Los re-INVITE también se pueden utilizar para poner una llamada que está en curso en espera. Para ello el UAC puede enviar una petición re-INVITE con el atributo SDP sendonly.

5.2.5. Routing en SIP

Como hemos visto, los campos **`From`** y **`To`** sirven para identificar a los extremos de un diálogo pero **NO** para encaminar mensajes. Para ello, se utilizan:

- En las peticiones la **`request-URI`**, que puede ir cambiando a lo largo del camino, y las cabeceras **`route`**.
- En las respuestas la pila de cabeceras **`via`**. Dado que cada dispositivo que origina o envía un mensaje SIP incluye su cabecera **`Via`**, éstas aseguran que las respuestas seguirán el mismo camino (a nivel SIP) que las peticiones.

Existen dos métodos de routing en función de como se traten las cabeceras **`request-URI`** y **`route`**:

- **Enrutamiento estricto SIP**:

  Se reescribe la Request-URI con el valor de la cabecera **`Route`** superior, ésta se elimina y se reenvía el mensaje a la nueva Request-URI. Es decir, la Request-URI siempre contiene la URI del siguiente salto que puede ser otro servidor o el UA final.

  Como se reescribe la Request-URI es necesario guardar la original en la última cabecera **`Route`** para poder recuperarla en el salto final.

  Este tipo de routing se considera **obsoleto**.

- **Enrutamiento laxo (loose) SIP**:

  El campo Request-URI no se modifica en los saltos SIP. En su lugar, el siguiente salto viene determinado por la cabecera **`Route`** superior.

  Implica un cambio significativo dado que la Request-URI no indica necesariamente donde se va a enviar la petición y ahora la última cabecera **`Route`** contendrá los datos del último salto en el camino, no del UA destino.

  Se indica que un equipo soporta loose SIP routing con el parámetro `lr`:

```
Record-Route: <sip:10.31.101.50;lr>
Route: <sip:172.20.120.10;lr>
```

5.3. Descubrimiento dinámico a través de DNS (DDDS)

Una de las principales funciones de SIP es que permite a un usuario registrarse en diferentes localizaciones.

Para hacer posible la comunicación en este escenario, el usuario debe, como mínimo, estar en un dominio que disponga de un inbound proxy y un servidor registrar y el llamante debe disponer de la AoR del usuario llamado. Por ejemplo: *sip:alice@example.com*.

Aunque se den las condiciones anteriores, aún quedan **cuestiones importantes por resolver...**

1. *¿Cual es la dirección IP a la que debemos enviar las peticiones para este usuario (e.j INVITE)?*
2. *¿Que protocolo o protocolos de transporte (UDP, TCP, SCTP, TLS, etc.) se pueden utilizar?*
3. *¿Cual es el puerto o puertos a los que debemos enviar las peticiones SIP?*

En un caso más extremo podríamos disponer de una URI no SIP, como por ejemplo *alice@example.com*. Entonces, además, podríamos preguntarnos:

4. *¿Bajo esta dirección, hay un usuario que soporta SIP?*

Para resolver las preguntas anteriores, podemos fijarnos en como se resuelven las URIs en Internet, por ejemplo, un servicio WEB.

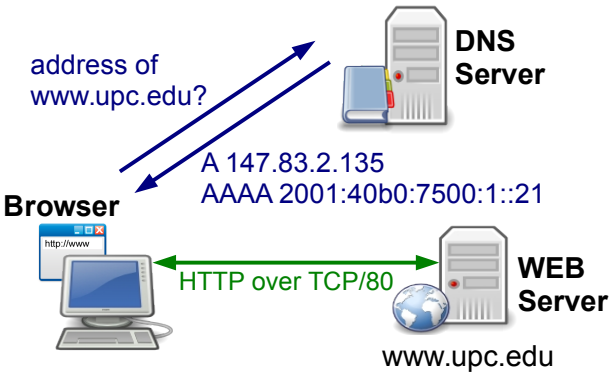


Figura 5.18: Resolución DNS de un servicio web.

Como ilustra la figura 5.18, la resolución de la URI http://www.upc.edu comporta los siguientes pasos:

- Se realiza una consulta al servicio DNS utilizando el nombre FQDN (Fully Qualified Domain Name) www.upc.edu para averiguar su dirección IP.
- El DNS devuelve la o direcciones correspondientes mediante uno o más registros de tipo A (Address, IPv4) y AAAA (IPv6).
- El cliente selecciona una de las direcciones en función de sus capacidades IP.
- Se extrapola que el protocolo de transporte será TCP y el puerto de servicio el 80.

En este ejemplo, el cliente ha deducido que el puerto de servicio es el 80 y el protocolo de transporte es TCP porque así lo dictamina el protocolo HTTP.

SIP, sin embargo, aunque si define un puerto por defecto, 5060, no determina cual es el protocolo de la capa de transporte por lo que un registro de tipo A o AAAA es insuficiente para descubrir el servicio.

El registro NAPTR se utiliza para informar acerca de los protocolos de transporte permitidos mientras que el registro SRV informa acerca de los puertos asociados a esos protocolos de transporte.

5.3.1. Registro SRV

El RFC2782[3] define un nuevo tipo de registro DNS para dotar de mayor flexibilidad al proceso de resolución de URIs.

Dado un servicio y un protocolo de la capa de transporte, el registro **SRV**[1] (SeRVice) proporciona y define el método de selección de FQDNs asociados incluyendo un esquema de reparto de carga.

La definición del registro SRV es la siguiente:

```
_service._protocol.name TTL class SRV priority weight port destination
```

- **Priority**: Indica la prioridad del FQDN. Un cliente debe seleccionar, en primer lugar, el de menor prioridad.
- **Weight**: Asigna pesos entre los registros con la misma prioridad. Un peso más grande significa que el candidato debe ser seleccionado con mayor probabilidad.
- **Port**: Indica el puerto del protocolo de transporte.
- **Destination**. Es el FQDN del servidor que presta el servicio.

Ejemplo de registros SRV:

```
_sip._tcp.example.com. 86400 IN SRV 10 60 15060 sipserver1.example.com.
_sip._tcp.example.com. 86400 IN SRV 10 40 5060  sipserver2.example.com.
```

[1] *Nota. Los registros SRV no se suelen utilizar en servicios "legacy" como web o e-mail.*

Gracias a este nuevo registro, es posible determinar a través de DNS si existe y cual es la IP y puerto de servicio asociados a una URI y un protocolo de transporte.

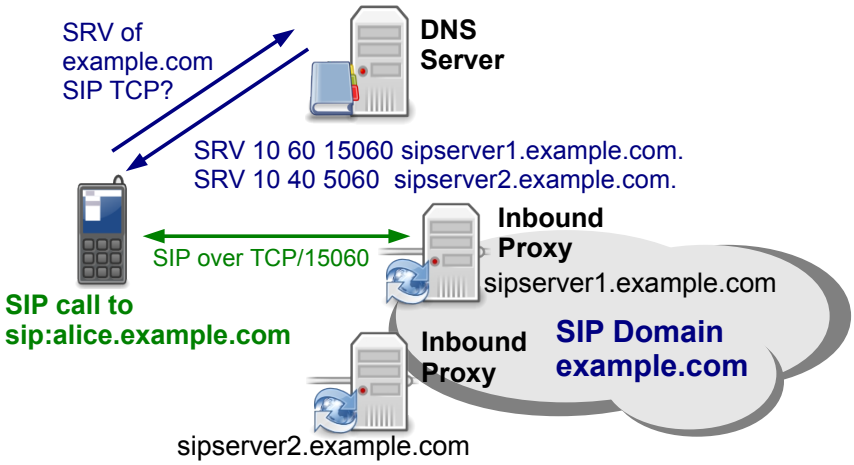


Figura 5.19: Resolución DNS de un servicio web.

5.3.2. Registro NAPTR

SIP no define ningún protocolo de transporte por defecto lo que obligaría a los clientes a ir realizando consultas DNS sobre los diferentes protocolos hasta encontrar el correcto si sólo existieran los registros SRV.

Con el fin de proporcionar un mecanismo aún más flexible, el RFC3403[4] define el **Dynamic Delegation Discovery System (DDDS)** en el que, como se menciona en el RFC3263[7], SIP se apoya:

"*The Session Initiation Protocol (SIP) uses DNS procedures to allow a client to resolve a SIP Uniform Resource Identifier (URI) into the IP address, port, and transport protocol of the next hop to contact. It also uses DNS to allow a server to send a response to a backup client if the primary client has failed.*"

DDDS, además de apoyarse en el uso de registros SRV, A y AAAA, define un nuevo registro denominado **Name Authority Pointer o NAPTR** que puede proporcionar:

1. Un nombre FQDN.
2. Un registro SRV con el que obtener un posible reparto de carga por servicio y protocolo de transporte.
3. Una nueva URI con la que volver a realizar otra consulta.
4. Una nueva URI modificada a partir de la original con una expresión regular con la que volver a realizar otra consulta.

El formato del NAPTR es el siguiente:

```
owner-name ttl class NAPTR order preference flag params regexp replace
```

Donde **`owner-name`**, **`ttl`** y **`class`** tienen el mismo significado que en el resto de registros DNS. Los demás parámetros responden a:

- **`Order`** (16 bits): Entero que especifica el orden en el que los registros NAPTR deben ser procesados, de menor a mayor.
- **`Preference`** (16 bits): Entero que especifica el orden en el que se deben priorizar los registros NAPTR con el mismo valor order. También de menor a mayor.
- **`Flag`**: Los valores son específicos para cada aplicación. En SIP se utilizan los siguientes:
  - **u**: Indica que en el campo **`regexp`** hay una URI válida.
  - **s**: Indica que en el campo **`replace`** hay el FQDN de un SRV.
  - **a**: Indica que en el campo **`replace`** hay el FQDN de un registro A (o AAAA).
  - **" "** (cadena vacía): Definida por ENUM para indicar que el campo **`regexp`** está vacío y que el campo **`replace`** contiene el FQDN de un nuevo NAPTR.

- **`Params`**: Define el tipo de aplicación y transporte. El formato es *protocolo+rs*.
  - *protocolo*: protocolo utilizado por la aplicación DDDS.
  - *rs* (*resolution service*): 0 o más *rs* separados por +. Indica el protocolo de capa inferior (transporte).
  - Ejemplos para SIP:

```
"SIP+D2T": SIP sobre TCP
"SIP+D2U": SIP sobre UDP
"SIPS+D2T": SIP con TLS sobre TCP
"SIP+D2S": SIP sobre SCTP
```

- **`Regexp`**: Se presenta entre comillas (“ ”) y contiene una expresión regular de sustitución que se aplica a la cadena original con el fin de construir el siguiente nombre de dominio a consultar. Sólo deben ser aplicados a la cadena original, nunca al resultado de un NAPTR anterior.
- **`Replace`**: Es una cadena que contiene el FQDN a consultar según lo que indique el campo **`flag`**.

Una vez se han recibido los registros NAPTR asociados a un dominio, los que tienen **`flags`** desconocidos o inapropiados se descartan y los registros restantes son ordenados por su campo **`order`**.

Dentro de cada valor de **`order`**, los registros se ordenan además por el campo **`preference`** y se procesan hasta que se encuentre un registro que haga coincidencia. Esto es:

- Tiene un valor en el campo **`replace`** en vez de en el campo **`regexp`**.
- El campo **`regexp`** hace coincidencia con la cadena solicitada. Es decir, se puede aplicar la expresión regular de sustitución.

Cuando un cliente encuentra un registro NAPTR con un servicio aceptable a su petición, no debe seguir analizando otros registros. Ello permite a los administradores dirigir a los clientes hacia protocolos de transporte más eficientes y / o seguros.

El NAPTR es una parte de DDDS y su funcionalidad sólo puede entenderse verdaderamente cuando se lee conjuntamente con la especificación detallada de la aplicación que lo utiliza. En el caso de SIP, el RFC326[7] recomienda que cuando se inicia un contacto con un usuario se haga una consulta a DNS por el registro **NAPTR** utilizando el dominio base.

Así, por ejemplo si se quiere contactar con sip:alice@example.com, se debe realizar inicialmente una consulta de los registros NAPTR asociados al dominio example.com y, sólo si ésta falla, se podrían hacer consultas directas de registros SRV.

La razón es que NAPTR permite definir a los administradores la preferencia en los servicios a ser utilizados mientras que los SRVs sólo proveen de detalles operacionales para un servicio como las direcciones IPs de los servidores, puertos y reparto de carga.

Por ejemplo, los siguientes registros NAPTR podrían utilizarse para definir el acceso a los inbound proxies del dominio example.com:

```
example.com. IN NAPTR 10 10 "s" "SIPS+D2T" "" _sips._tcp.example.com.
example.com. IN NAPTR 20 10 "s" "SIP+D2T"  ""  _sip._tcp.example.com.
example.com. IN NAPTR 30 10 "s" "SIP+D2U"  ""  _sip._udp.example.com.
```

Esencialmente los registros previos indican que si el usuario soporta SIP sobre TLS debe utilizar este servicio ya que es el prioritario (order 10). Si no, SIP sobre TCP (order 20) y, por último, SIP sobre UDP (order 30).

Finalmente, los parámetros específicos de cada protocolo de transporte vendrán definidos en los registros SRV correspondientes.

Así, la configuración DNS de este ejemplo podría ser la siguiente:

```
example.com IN SOA ns.example.com. root.example.com. (
          2003032001
          10800
          3600
          604800
          86400 )

example.com.              43200 IN NS      ns.example.com.
;
ns.example.com.           43200 IN A       10.0.0.20
sipserver1.example.com.   43200 IN A       10.0.0.21
sipserver2.example.com.   43200 IN A       10.0.0.22
;
_sip._udp.example.com.    43200 IN SRV 0 0 5060  sipserver1.example.com.
_sip._udp.example.com.    43200 IN SRV 1 0 5060  sipserver2.example.com.
_sip._tcp.example.com.    43200 IN SRV 0 4 5060  sipserver1.example.com.
_sip._tcp.example.com.    43200 IN SRV 0 2 5060  sipserver2.example.com.
_sips._tcp.example.com.   43200 IN SRV 0 0 5060  sipserver1.example.com.
_sips._tcp.example.com.   43200 IN SRV 0 0 5060  sipserver2.example.com.
;
example.com. IN NAPTR 0 0 "s" "SIPS+D2T" "" _sips._tcp.example.com.
example.com. IN NAPTR 1 0 "s" "SIP+D2T"  "" _sip._tcp.example.com.
example.com. IN NAPTR 2 0 "s" "SIP+D2U"  "" _sip._udp.example.com.
```

5.3.3. E.164 & ENUM

Una de las cuestiones aún no resueltas es como gestionar una llamada desde un dominio SIP a un usuario en una red de telefonía convencional. En este caso, el usuario utiliza la numeración telefónica definida en el estándar E.164 y sigue el típico formato +5116262000 o también se puede expresar como tel:+5116262000.

En realidad, la numeración E.164 puede identificar a un usuario en una red de circuitos pero también a un usuario en una red SIP.

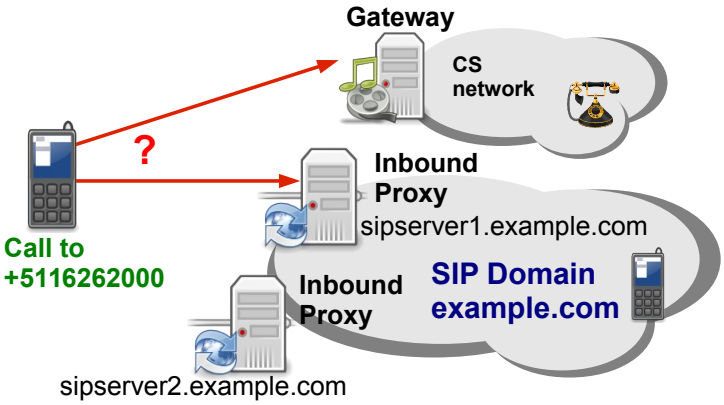


Figura 5.20: Disyuntiva en la resolución de numeración e.164.

Surge la necesidad de definir una forma de transformar números E.164 a nombres FQDN y disponer así de DDDS para descubrir el método y la localización de los dispositivos que permiten iniciar la comunicación.

El grupo de trabajo ENUM (E.164 Number to URI Mapping) del IETF ha especificado la norma para realizar esta transformación. ENUM traduce los números de teléfono en direcciones de Internet que pueden contener una referencia a una URI SIP, otro número de teléfono, una página web o una dirección de correo electrónico.

Los números de teléfono se transforman mediante la inversión de los dígitos a partir del código de marcación internacional (1 = EE.UU. / Canadá, 44 = U.K., 34 = España, etc.) y la inserción de puntos entre todos ellos. Finalmente, se añade el dominio e164.arpa. Por ejemplo la transformación del número de teléfono 123456789 de España (34) sería “9.8.7.6.5.4.3.2.1.4.3.e164.arpa”.

De esta forma, un servidor con soporte ENUM puede realizar primero una consulta de los registros NAPTR asociados a un número de teléfono en el árbol de DNS para descubrir si hay formas alternativas de establecer la llamada en lugar de simplemente llamando a la línea telefónica PSTN.

ENUM define como utilizar el campo **flag** de forma específica:

- El campo **flag** sólo utiliza los valores *u* y " ".
- El flag *u* indica que el resultado de aplicar el campo **regexp** será una URI válida para el protocolo definido en **params** y el campo **replace** debe ignorarse.
- El flag vacío ("") indica que el campo **replace** contiene el FQDN para una nueva búsqueda NAPTR.

El campo **params** utiliza también una variante de la definición estándar. Este campo debe comenzar con el valor E2U (E.164 a URI). La forma genérica de este campo utilizado por ENUM es:

```
E2U + enumservice [+ enumservice]
enumservice = tipo [: subtipo]
```

Los campos de tipo y subtipo se definen en la lista mantenida por el grupo IANA Enumservice. Ejemplos de valores válidos de esta lista son:

```
E2U+pres
E2U+voice:tel+sms:tel (forma combinada)
E2U+web:http
E2U+sms:mailto
E2U+sms:tel
E2U+sip
E2U+pstn
E2U+tel
```

Además ENUM permite una definición especial de tipos:

- Si el tipo empieza por 'X-" indica que se trata de un servicio experimental. Por ejemplo "X-TSIP" podría indicar una extensión SIP experimental.
- Si el tipo empieza por 'P-" indica que se trata de un servicio privado. Por ejemplo, "p-mms" podría indicar un servicio multimedia privado. Los servicios privados pretenden tener un alcance local, es decir, sólo deben utilizarse dentro de una red privada y nunca debe aparecer de forma visibles en la Internet pública (global).

Los valores alternativos a sip del campo "enumservice" permiten informar de métodos de contacto adicionales. Por ejemplo, los registros NAPTR que se muestran a continuación informan de que si no es posible el contacto mediante SIP, se puede utilizar email:

```
$ORIGIN 6.2.6.1.1.5.e164.arpa.
0.0.0.2 IN NAPTR 10 10 "u" "E2U+sip"    "!^.*$!sip:info@example.org!i" .
0.0.0.2 IN NAPTR 10 20 "u" "E2U+mailto" "!^.*$!mailto:info@example.org!i" .
```

ENUM permite la delegación del dominio e164.arpa de forma análoga a la de cualquier otro dominio DNS.

En europa, el dominio "e164.arpa" está gestionado por RIPE que delega los subdominios a las autoridades de cada país aunque no todos los países han solicitado esta delegación. Podemos descubrir fácilmente cuáles han sido delegados al consultar sus registros NS. Por ejemplo, los servidores DNS autorizados para el Reino Unido son:

```
:~$ dig  NS 4.4.e164.arpa
...
;; ANSWER SECTION:
4.4.e164.arpa.          172620  IN      NS      nsc.nic.uk.
4.4.e164.arpa.          172620  IN      NS      nsa.nic.uk.
4.4.e164.arpa.          172620  IN      NS      nsd.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns1.nic.uk.
4.4.e164.arpa.          172620  IN      NS      nsb.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns2.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns7.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns4.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns6.nic.uk.
4.4.e164.arpa.          172620  IN      NS      ns5.nic.uk.

;; ADDITIONAL SECTION:
nsc.nic.uk.             172620  IN      A       156.154.102.3
```

Sin embargo, si consultamos la delegación de la numeración de España, veremos que ésta no ha sido solicitada, lo que significa que el servicio ENUM no se encuentra disponible.

```
:~$ dig  NS 4.3.e164.arpa

;; QUESTION SECTION:
;4.3.e164.arpa.                 IN      NS

;; AUTHORITY SECTION:
e164.arpa.              3600    IN      SOA     pri.authdns.ripe.net. dns.ripe.net.
1429185659 3600 600 864000 7200
```

5.3.4. NRENUM

NRENum es una iniciativa surgida de la red académica europea (GEANT) con el fin de permitir llamadas VoIP a otros sistemas VoIP directamente a través de Internet en los países donde ENUM (e164.arpa.) no se encuentra disponible. Para ello, la asociación GÉANT (anteriormente conocida como TERENA) creó el arbol de delegación "nrenum.net".

GEANT coordina el dominio "nrenum.net" y lo delega a las diversas organizaciones NREN (National Research and Education Networking) de los países europeos.

Aunque el propósito principal era su uso dentro la red académica, las NREN ofrecen el servicio también a empresas y usuarios de teléfonos regulares en países que no cuentan con la delegación de ENUM.

En el caso de España, la delegación está operada por RedIRIS:

```
:~$ dig  NS 4.3.nrenum.net

; <<>> DiG 9.9.5-3ubuntu0.8-Ubuntu <<>> NS 4.3.nrenum.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14464
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;4.3.nrenum.net.                        IN      NS

;; ANSWER SECTION:
4.3.nrenum.net.         172800  IN      NS      sun.rediris.es.
4.3.nrenum.net.         172800  IN      NS      chico.rediris.es.

;; ADDITIONAL SECTION:
sun.rediris.es.         15267   IN      A       130.206.1.2
sun.rediris.es.         1724    IN      AAAA    2001:720:418:caf1::2
chico.rediris.es.       15107   IN      A       130.206.1.3
chico.rediris.es.       1530    IN      AAAA    2001:720:418:caf1::3
```

5.4. NAT y VoIP

Los elementos de red que hacen NAT (Network Address Translation) provocan que muchas aplicaciones IP dejen de funcionar si no se corrigen sus efectos.

En particular NAT afecta al correcto funcionamiento de las comunicaciones mediante SIP y RTP. El problema radica en que en estos dos protocolos se hace referencia explícita a la dirección IP y puerto habilitado por el cliente tanto para el flujo media como para el tráfico de señalización. El comportamiento por defecto de NAT es modificar direcciones a nivel IP (capa 3 del modelo OSI) y puertos a nivel de transporte (capa 4 del modelo OSI) pero no examina ni modifica el contenido de los mensajes que hacen referencia a estos parámetros. Como consecuencia, las IPs y puertos referenciados quedan inaccesibles.

5.4.1. Tipos de NAT

Full Cone

En el NAT tipo "Full Cone", véase figura 5.21, todas las peticiones desde la misma IP interna son traducidas a una misma única IP externa mientras que el puerto no se modifica. Gracias a esta relación 1 a 1, un host des de la red pública puede enviar paquetes al host interno, simplemente enviando el paquete a la IP y puerto externos.

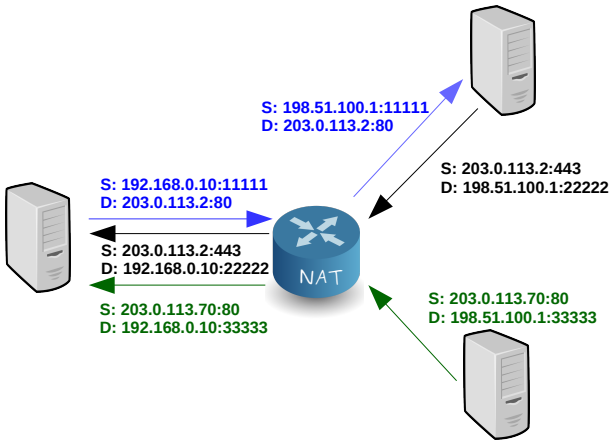


Figura 5.21: Full Cone.

Restricted Cone

En el NAT tipo "Restricted Cone", véase figura 5.22, todas las peticiones desde la misma IP y puerto interno son mapeados a la misma IP externa y puerto externo.

Al contrario que un full-cone, un host externo puede enviar un paquete al host interno sólo si el host interno ha mandado previamente un paquete a la IP de dicho host externo.

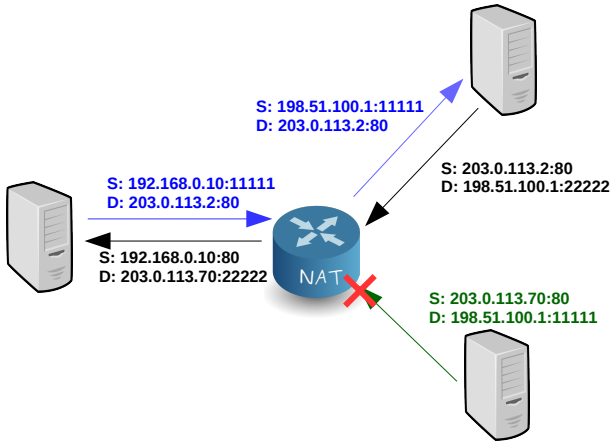


Figura 5.22: Restricted Cone.

Port Restricted Cone

El tipo "Port Restricted Cone", véase figura 5.23, es similar al restricted cone pero la restricción llega al nivel de puerto. Es decir, un host externo puede enviar un paquete a un host interno siempre y cuando el host interno haya enviado previamente un paquete a la IP y puerto de dicho host externo.

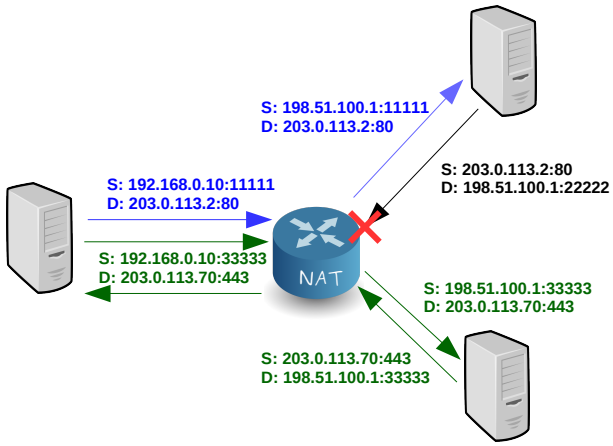


Figura 5.23: Restricted Cone.

Symmetric NAT

Todas las peticiones desde la misma IP y puerto internos dirigidas a una IP y puerto de la red pública son mapeados a una misma IP/Puerto externo que pueden tomar valores aleatorios.

A diferencia del Port Restricted Cone, la traducción IP/Puerto privado a IP/Puerto público se mantiene únicamente durante un cierto tiempo y no se conserva en sucesivas peticiones después de haber expirado. Es decir, por cada nueva comunicación saliente se asigna un puerto aleatorio, véase figura 5.24. Sólo el host externo que recibe el paquete puede enviar paquetes de vuelta al host interno.

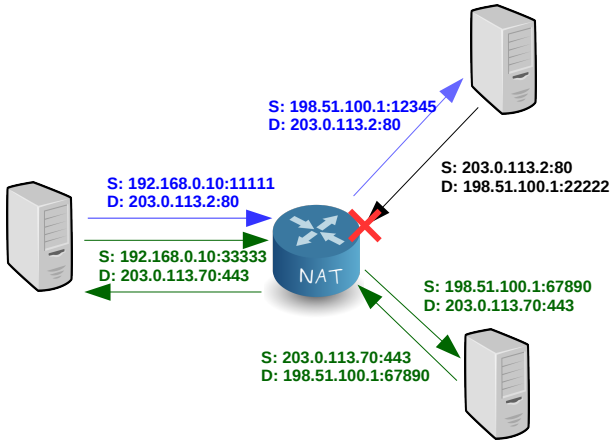


Figura 5.24: Symmetric NAT.

Hairpinning en NAT

Cuando un dispositivo NAT es capaz de reenviar paquetes de un host en una red interna a otro host en la misma red interna se dice que implementa "hairpinning".

Para ello, detecta que la IP pública destino corresponde con una asociación NAT ya creada a una dirección IP de un host interno. Es un comportamiento deseable pero no todos los dispositivos lo soportan. Un router NAT

sin hairpinning no posibilitará que dos hosts internos detrás de un mismo NAT se puedan comunicar usando sus direcciones externas.

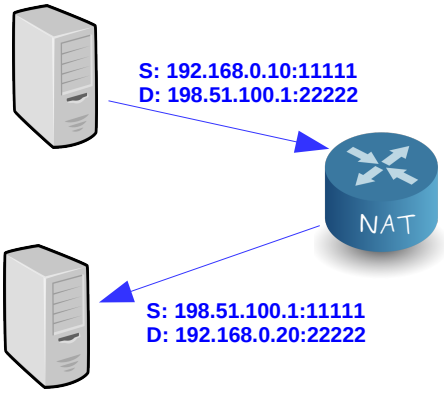


Figura 5.25: Hairpinning.

5.4.2. NAT y la señalización SIP

En este apartado analizaremos en detalle los obstáculos que introduce NAT en la comunicación VoIP sobre SIP y RTP.

Peticiones SIP desde la red interna

El primer problema que plantea NAT es en el envío de peticiones SIP sobre UDP de un UA que está detrás de un NAT a un UA en la red externa.

SIP sobre UDP envía las respuestas a la IP origen del paquete (capa 3) y el puerto que se indica en el header **`Via`**.

Dado que los dispositivos NAT no reescriben las cabeceras SIP, como vemos en el ejemplo de la figura 5.26, el puerto informado en el header **`Via`** de la petición una vez ha atravesado este dispositivo sigue siendo el puerto del UA interno, no el puerto abierto (pin hole) en el NAT.

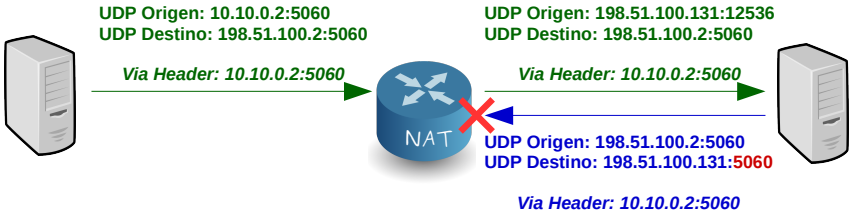


Figura 5.26: Header Via a través de NAT.

Como consecuencia, las peticiones SIP/UDP que atraviesan un dispositivo NAT pueden dar lugar a respuestas que se envían a un puerto que no corresponde a un pin hole en el NAT por lo que éstas son descartadas.

Una solución sería utilizar SIP sobre TCP dado que en esta implementación las respuestas SIP se envían a la dirección IP y puerto del paquete origen, no a la información indicada en las cabeceras SIP. Desafortunadamente, no todos los dispositivos SIP soportan TCP y SIP sobre UDP sigue siendo una implementación extendida por lo que es necesario un método alternativo.

Es por ello que en el RFC3581[8] se definió un nuevo parámetro, `rport`, para el header **`Via`**. El UAC añade este parámetro sin ningún valor para indicar que soporta esta extensión:

```
REGISTER sip:198.51.100.2 SIP/2.0
Via: SIP/2.0/UDP 10.10.0.2:5060;rport;
branch=z9hG4bKPjnRa-EXjBjlcE7T94v-fHqLwpM6q2nZjT
Route: <sip:198.51.100.2;lr>
...
Contact: <sip:bob@10.10.0.2:5060;ob>;+sip.ice
...
```

El UAS, al procesar la petición, si observa que el campo **`Via`** lo contiene, notifica al UAC del puerto a nivel de transporte observado asignándolo al parámetro `rport` e incluyéndolo en el **`Via`** de la respuesta. Además el UAS enviará la respuesta a este puerto:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.10.0.2:5060;rport=12536;
branch=z9hG4bKPjnRa-EXjBjlcE7T94v-fHqLwpM6q2nZjT;received=198.51.100.131
...
```

Así, si la asociación NAT correspondiente no ha expirado, la respuesta llegará al UAC sin problemas y éste podrá utilizar el valor de `rport` como puerto de contacto en sucesivas peticiones:

```
REGISTER sip:198.51.100.2 SIP/2.0
Via: SIP/2.0/UDP 198.51.100.131:12536;rport;
branch=z9hG4bKPjaySx27bGtyZGWpeeDIvCKHIA5emWjBaW
Route: <sip:198.51.100.2;lr>
...
Contact: <sip:bob@198.51.100.131:12536;ob>;+sip.ice
...
```

Peticiones desde la Red Externa

Ahora consideremos que sucede si la petición SIP proviene de la red externa hacia un UA detrás de un NAT.

En este caso, únicamente un dispositivo que implemente nat de tipo Full-Cone permitirá que las peticiones lleguen al cliente interno. Ante cualquier otro tipo de NAT, la comunicación no es viable salvo que el cliente de la red interna sea el que inicie la comunicación.

Es decir, un UAC detrás de un NAT para poder recibir peticiones debe haber iniciado previamente una comunicación hacia la red externa y si además hablamos de nat simétrico, ésta deberá ser con el host del que espere recibir tráfico.

Para ello se hace uso de un servidor en la red pública que hará las funciones de inbound y outbound proxy y un servidor registrar.

Durante el proceso de registro el cliente indica la dirección en la que desea recibir solicitudes SIP entrantes (Address of Record o AOR):

```
REGISTER sip:198.51.100.2 SIP/2.0
Via: SIP/2.0/UDP 10.10.0.2:5060;rport;
branch=z9hG4bKPjnRa-EXjBjlcE7T94v-fHqLwpM6q2nZjT
Route: <sip:198.51.100.2;lr>
...
CSeq: 18283 REGISTER
Contact: <sip:bob@10.10.0.2:5060;ob>;+sip.ice
...
```

Mediante el registro, el UA no sólo consigue abrir un pin hole en el NAT si no que es además capaz de conocer su dirección y puerto en la red externa gracias a los campos **received** y **rport** que recibe en la respuesta del servidor:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.10.0.2:5060;rport=12536;
branch=z9hG4bKPjnRa-EXjBjlcE7T94v-fHqLwpM6q2nZjT;received=198.51.100.131
...
CSeq: 18283 REGISTER
...
```

El UA puede entonces enviar una segunda petición de registro utilizando estos datos para corregir los datos relativos a su localización.

```
REGISTER sip:198.51.100.2 SIP/2.0
Via: SIP/2.0/UDP 198.51.100.131:12536;rport;
branch=z9hG4bKPjaySx27bGtyZGWpeeDIvCKHIA5emWjBaW
Route: <sip:198.51.100.2;lr>
...
CSeq: 18284 REGISTER
Contact: <sip:bob@198.51.100.131:12536;ob>;+sip.ice
...
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 198.51.100.131:12536;rport=12536;
branch=z9hG4bKPjaySx27bGtyZGWpeeDIvCKHIA5emWjBaW;received=198.51.100.131
...
CSeq: 18284 REGISTER
Contact: <sip:bob@198.51.100.131:12536;ob>;expires=300
...
```

Por último, dado que el proceso de registro se ha llevado a cabo mediante un proxy SIP, éste será el host habilitado en el nat para enviar peticiones al UA interno.

Binding Timeout

Las asociaciones NAT tienen un tiempo de vida en inactividad determinadas por un temporizador que en cada protocolo (TCP, UDP, etc.) es diferente. Estos tiempos varían en función del fabricante pero generalmente el tiempo de vida para una conexión UDP suele ser del orden de minuto(s).

Al alcanzar el periodo máximo, la asociación nat se destruye por lo que el cliente debe mantener viva esa asociación generando periódicamente actividad. Para ello se contemplan varios métodos denominados "keep-alive": como el envío de mensajes SIP periódicos con "CR+LF" al estilo de heartbeat o el uso de protocolos externos como STUN que se verá más adelante.

5.4.3. SDP y negociación del RTP

Cuando se atraviesan dispositivos NAT, el establecimiento del flujo multimedia (RTP) introduce un nuevo reto. Aunque RTP define puertos por defecto, muchos clientes utilizan puertos aleatorios y el canal se negocia "a ciegas" mediante SDP dentro del cuerpo de los mensajes SIP.

Mientras que, como hemos visto, SIP si que dispone de métodos intrínsecos al protocolo para corregir la información, SDP no. Así, en el siguiente ejemplo podemos ver que, mientas la información relativa a la IP y puerto en las cabeceras SIP se ha corregido, el contenido de SDP es incorrecto.

Ejemplo de cabeceras SIP de un UA interno que ha corregido la información de su IP y puerto en la red pública gracias a los parámetros `rport` y `received`:

```
 INVITE sip:claire@198.51.100.132:5060 SIP/2.0
Via: SIP/2.0/UDP
198.51.100.130:5060;rport;branch=z9hG4bKPj4.wF8QQ79EQuhgT71yHND49myCTs-Rdr
Max-Forwards: 70
From: sip:alice@example.org;tag=dSoEDWbFVDr0C-jle0lQUXjX.-GBiGvq
To: sip:claire@198.51.100.132
Contact: <sip:alice@198.51.100.130:5060;ob>
```

Contenido del cuerpo (SDP) de ese mismo mensaje SIP:

```
o=- 3644248265 3644248265 IN IP4 192.168.0.2
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 4000 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 192.168.0.2
b=TIAS:64000
a=rtcp:4001 IN IP4 192.168.0.2
...
```

Los clientes por tanto, a priori, no disponen de ninguna forma de conocer la dirección IP y puerto que observará el otro extremo antes de que se inicie la comunicación RTP. De hecho, aún no se habrán ni abierto los pin holes correspondientes en el NAT.

A lo largo de los años se han desarrollado varias soluciones, algunas parciales, que veremos en detalle:

- Soluciones dependientes del cliente SIP:
  - Symmetric RTP detection
  - Sesión Traversal Utilities for NAT (STUN)
  - Traversal Using Relays around NAT (TURN)
  - Interactive Connectivity Establishment (ICE)
- Soluciones independientes del cliente SIP:
  - Application Layer Gateways (ALG) y ALG con media GW
  - RTP-Proxy
  - UPnP y el protocolo IGD
  - Back to Back User Agent (B2BUA)

Symmetric RTP detection

Se trata de una técnica en la que el cliente hace caso omiso del campo contact de SDP si éste no coincide con el origen de los primeros paquetes RTP recibidos. El cliente modifica entonces el destino y envía sus paquetes a la dirección IP y puerto observados.

Garantiza que el tráfico RTP llegará al cliente interno ya que la asociación en el NAT ya se ha creado. Por contra, requiere que los clientes esperen recibir el tráfico en el mismo puerto que utilizan para el envío.

Además, esta solución sólo funciona si un cliente detrás de un NAT simétrico no intenta comunicarse con otro cliente detrás de otro NAT simétrico o un NAT de tipo port-restricted.

Sesión Traversal Utilities for NAT (STUN)

STUN es un protocolo sencillo de tipo cliente-servidor que permite detectar el par IP-puerto observados desde la red externa. A este tipo de direcciones se les llama "server-reflexive".

Básicamente, como se aprecia en la figura 5.27, el dispositivo interno realiza una petición a un servidor STUN y éste, en su respuesta, le informa de la IP y puerto origen observados. Es por ello que es imprescindible que el servidor se encuentre en la red pública o en la red de un proveedor de servicio.

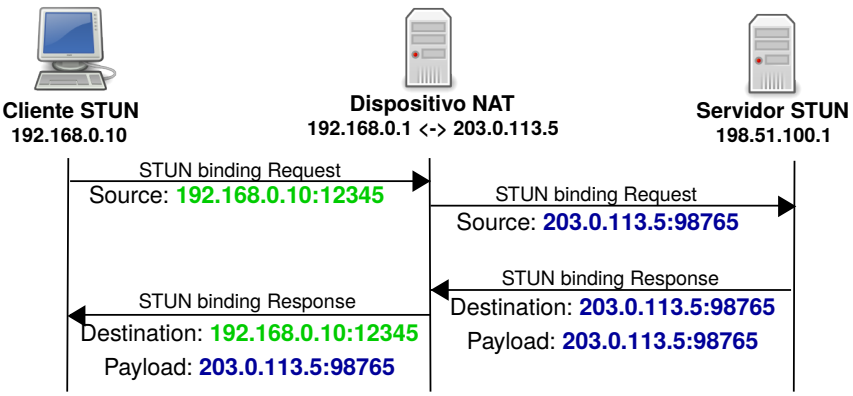


Figura 5.27: Flujo de comunicación del protocolo STUN.

Algunos dispositivos NAT tratan de ser inteligentes inspeccionando también el payload de los paquetes y cambiando todas las referencias a la dirección server-reflexive. Como consecuencia, una nueva versión de STUN fue necesaria en la que se ofusca la dirección contenida en el payload realizando un XOR con un valor conocido.

Cabe mencionar que STUN no sirve si un cliente está detrás de un NAT simétrico ya que la asociación sólo es válida para esa comunicación específica entre el cliente interno y el servidor STUN.

Traversal Using Relays around NAT (TURN)

TURN surge de la necesidad de crear un mecanismo alternativo cuando STUN es insuficiente, como en le caso de NAT simétrico.

En esencia se trata de encaminar el flujo multimedia a través de un servidor intermedio (relay) ubicado en la red pública.

Esto añade un coste considerable ya que, como podemos ver la figura 5.28, mientras que un servidor STUN sólo interviene en la consulta inicial de los clientes SIP, un servidor TURN hace de relay para todo el tráfico multimedia.

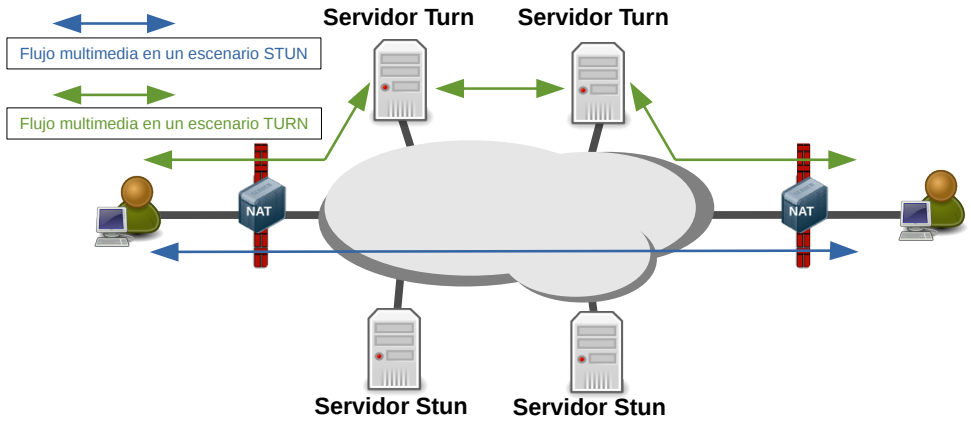


Figura 5.28: Flujo multimedia TURN vs STUN.

TURN utiliza el mismo formato en los mensajes que el protocolo STUN. El proceso para establecer el canal a través del relay TURN consta de dos fases:

1. Petición y asignación de recursos en el servidor TURN.
2. Activación del canal de comunicación.

En la primera fase el cliente solicita la reserva de recursos. Un servidor TURN puede aceptar peticiones anónimas o restringidas por usuario y contraseña. En el caso de aceptar peticiones anónimas, el servidor, al recibir la solicitud (Allocate request), reserva un puerto y responde con un "Allocate success". En el caso de ser necesaria autenticación, como se observa en el ejemplo de la figura 5.29, el servidor responde con un "Allocate error (401)" para indicar que el cliente debe volver a enviar la solicitud, esta vez incluyendo información de usuario y contraseña.

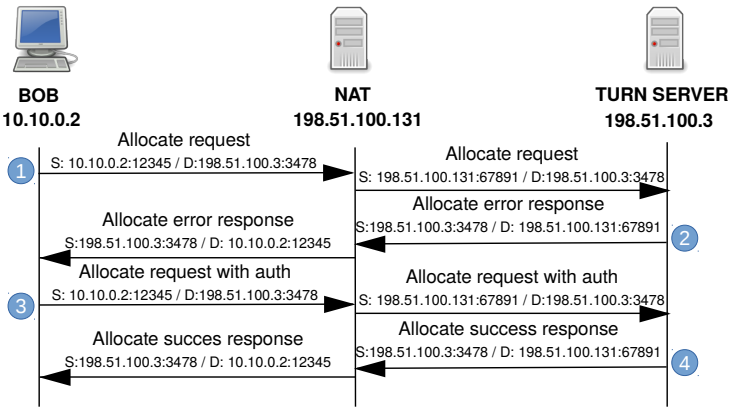


Figura 5.29: Fase 1 - Petición y reserva de recursos.

La información que se incluye en el cuerpo del "Allocate success" es la siguiente:

- XOR de la IP y puerto del cliente observados desde la red pública y un ID de la transacción.
- IP y puerto reservado en el servidor TURN.

En este punto, el cliente dispone de una dirección de transporte que puede utilizar para recibir el flujo multimedia. Sin embargo, como se observa en la figura 5.30, hasta que el cliente no negocia con el servidor TURN la activación de los recursos reservados (segunda fase), éste no permite la retransmisión de ningún dato.

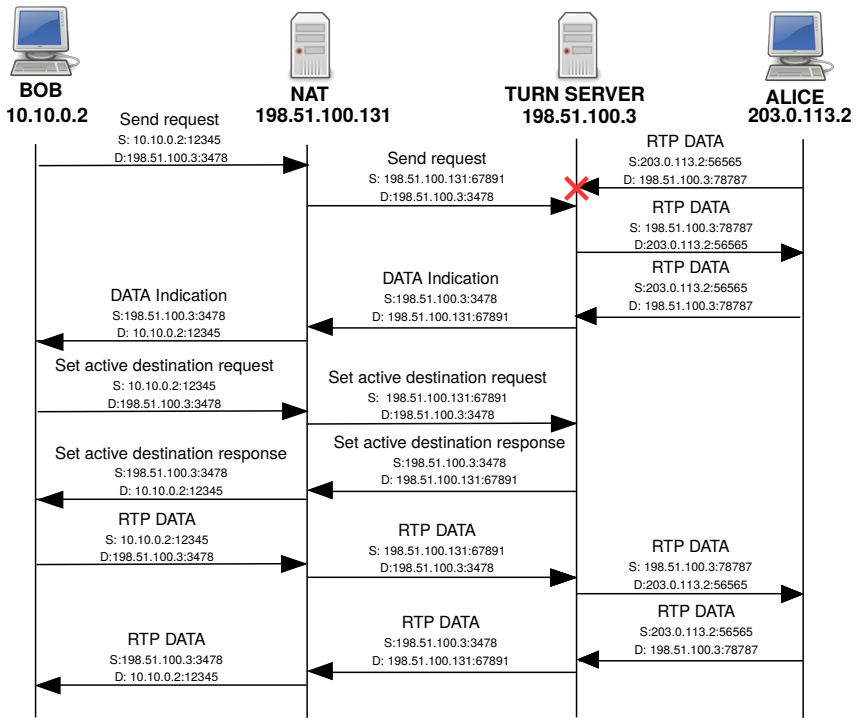


Figura 5.30: Fase 2 - Activación de recursos.

Cuando el primer paquete con destino el cliente llega al puerto asignado en el servidor TURN, éste comprueba si se disponen de permisos para su recepción y lo deniega en caso negativo. Estos permisos se establecen cuando el cliente, "Bob" en el ejemplo anterior, envía una solicitud de tipo **Send request** al servidor TURN incluyendo:

- Atributo Data: Contiene los datos a transmitir, en este caso RTP.
- Atributo Destination Address: La dirección y puerto del destinatario. "Alice" en nuestro ejemplo.

Al recibir esta información, el servidor TURN anota la dirección y puerto de Alice como permitida para recibir datos y activa el relay de datos en sentido Bob - Alice.

En este punto, al volver a recibir datos de Alice, origen ya conocido y con permisos de recepción, el servidor notifica de la recepción de datos a Bob enviándole un mensaje de tipo "Data indication" que contiene los siguientes atributos:

- Data: datos recibidos (RTP).
- Remote Address: dirección y puerto del origen (Alice).

Bob ahora puede solicitar la activación del relay de datos en sentido Alice - Bob mediante un mensaje de tipo "Set Active Destination Request". El servidor TURN, confirma con un "Set Active Destination response" quedando activo el canal de comunicación entre Bob y Alice a través del relay.

Interactive Connectivity Establishment (ICE)

ICE combina técnicas de STUN y TURN para encontrar la mejor ruta de comunicación con el otro extremo. Se basa en el intercambio de candidatos, combinación de dirección IP y puerto, con los que se negocia el camino, parejas de candidatos, óptimo para el flujo multimedia.

El proceso de negociación ICE consiste en varias etapas:

1. Descubrimiento de candidatos:

   Antes de enviar la invitación, el UA obtiene un listado de todos sus candidatos disponibles. Existen varios tipos de candidatos:

   - **Host candidates (host)**: Obtenidos de las tarjetas de red del propio host, incluyendo enlaces VPN, etc.
   - **Server-reflexive (srflx)**: Candidatos obtenidos al realizar consultas a un servidor STUN.
   - **Relay (relay)**: candidatos Obtenidos mediante la solicitud a un relay, por ejemplo TURN.
   - **Peer-reflexive (prflx)**: Candidatos del peer descubiertos durante la propia negociación de ICE.

2. Priorización de candidatos:

   Se aplican prioridades de manera que unos candidatos sean preferentes frente a otros. El cálculo de esta realiza según la siguiente fómula:

```
prioridad = (2^24)*(Tipo de candidato) +
            (2^8)*(Preferencia para cadidatos locales) +
            (2^0)*(256 - ID de componente)
```

   - **Tipo de candidato**: Valor numérico en función del tipo de candidato. Los valores van de 0, menos prioridad, a 126, máxima prioridad, siendo 0 el valor asignado al relay y 126 para los candidatos locales.
   - **Preferencia para candidatos locales**: Se trata de una política para la selección de candidatos del mismo tipo. También según preferencias IPv4/IPv6.
   - **ID de componente**: Valor fijado según el protocolo negociado. 1 para RTP, 2 para RTCP.

3. Codificación de candidatos:

   Los candidatos se codifican en SDP como se muestra a continuación y se envían al otro extremo para iniciar una negociación de tipo oferta - respuesta:

```
a=candidate:Rc6336403 1 UDP 16777215 198.51.100.3 57043 typ relay raddr 198.51.100.131 rport 4000
```

   - **Foundation** (*Rc6336403*): Identificador único para cada candidato del mismo tipo, misma interfaz y mismo servidor STUN/TURN si procede.
   - **Identificación de componentes (*1*)**: Valor fijado según el protocolo negociado. 1 para RTP, 2 para RTCP.
   - **Transporte** (*UDP*): Protocolo de transporte ofrecido por el candidato.
   - **Prioridad** (*16777215*): Prioridad del candidato.
   - **Dirección IP y puerto** (*198.51.100.3 55062*): IP y puerto del candidato.
   - **Tipo** (*typ relay*): Tipo de componente. Los posibles valores son: host, srflx, prflx y relay.
   - **Related Address** (*raddr 198.51.100.131 rport 4002*): Información opcional. En el caso del uso de un relay contiene la server-reflexive IP del servidor y para candidatos STUN contiene la dirección del host (es decir, la dirección IP privada).

   Los candidatos codificados se envían en el cuerpo SDP del INVITE como muestra el siguiente ejemplo:

```
...
a=ice-ufrag:3b8fda70
a=ice-pwd:435806ec
a=candidate:Ha0a0002 1 UDP 2130706431 10.10.0.2 4015 typ host
a=candidate:Rc6336403 1 UDP 16777215 198.51.100.3 57043 typ relay raddr 198.51.100.131 rport 4000
a=candidate:Ha0a0002 2 UDP 2130706430 10.10.0.2 4024 typ host
a=candidate:Rc6336403 2 UDP 16777214 198.51.100.3 52552 typ relay raddr 198.51.100.131 rport 4020
```

El destinatario al recibir las ofertas, comienza su propio proceso de selección de candidatos, priorización y codificación. Una vez concluido, envía sus candidatos en el 200 OK:

```
SIP/2.0 200 OK
...
a=ice-ufrag:16c45494
a=ice-pwd:08ced713
a=candidate:Hc0a80002 1 UDP 2130706431 192.168.0.2 4018 typ host
a=candidate:Rcb007103 1 UDP 16777215 203.0.113.3 60181 typ relay raddr 198.51.100.130 rport 4014
a=candidate:Hc0a80002 2 UDP 2130706430 192.168.0.2 4025 typ host
a=candidate:Rcb007103 2 UDP 16777214 203.0.113.3 60253 typ relay raddr 198.51.100.130 rport 4030
```

4. Asignación de candidatos:

   Ahora que ambos extremos conocen los candidatos disponibles, cada agente involucrado en la comunicación empareja sus candidatos con los remotos para formar parejas de candidatos (caminos).

   Se hace una primera purga de caminos duplicados y las parejas restantes serán evaluadas por orden de prioridad descendente. A continuación, cada extremo inicia un proceso de verificación para cada uno de los pares de candidatos.

5. Verificación de candidatos:

   Se realizan pruebas de conectividad cada 20ms de cada uno de los caminos posibles mediante paquetes especiales STUN que contienen binding requests y aquellas combinaciones que reciben respuesta se confirman como candidatos validados. La figura 5.31 muestra un ejemplo del flujo de estos mensajes:

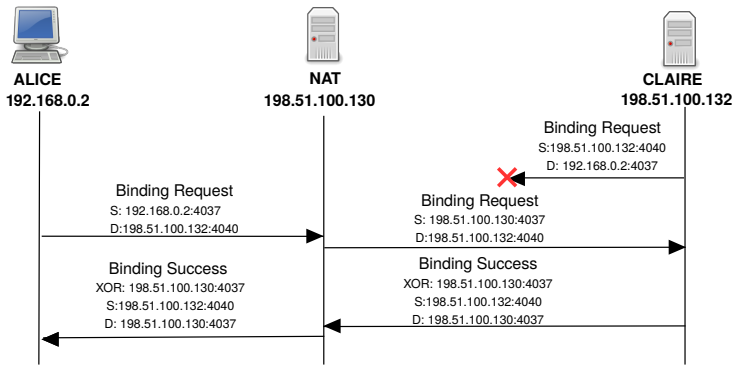


Figura 5.31: ICE - Verificación de candidatos.

6. Coordinación de candidatos:

   Tras la negociación, ambos agentes involucrados han de terminar con un par de candidatos válidos por cada componente (RTP y RTCP). El agente controlador, habitualmente el que realiza la llamada, nominará la pareja de candidatos validados de mayor prioridad por cada componente y volverá a validar estos caminos mediante otro binding request de STUN. Esta vez, como muestra el ejemplo de la figura 5.32, se incluye un atributo adicional, USE-CANDIDATE. Finalmente, ambos agentes utilizarán el par de candidatos que ha pasado las pruebas de conectividad y que además esté nominado.

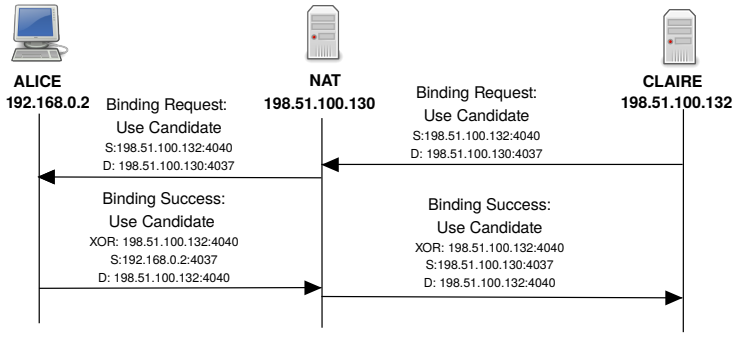


Figura 5.32: ICE - Coordinación de candidatos.

7. Confirmación:

   Aunque toda la negociación ha tenido lugar entre los UA finales es posible que haya otros agentes en el medio de la señalización, como por ejemplo proxys. Para que los proxys o las middle-boxes entre el llamado y el llamante estén al tanto de lo sucedido, se envía un re-INVITE o un UPDATE con el resultado de la negociación en el caso de que el candidato seleccionado no sea el candidato por defecto. Es decir, en el caso que no coincida con el que figuraba en las líneas "c" y "m" del SDP. Notar que estas lineas también se modifican con la información actualizada ya que son las que recogerán tanto los dispositivos qu hablan ICE como los que no.

```
UPDATE sip:alice@198.51.100.130:5060;ob SIP/2.0
...
Content-Type: application/sdp
...
o=- 3655999059 3655999060 IN IP4 198.51.100.3
...
m=audio 57043 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 198.51.100.3
a=rtcp:52552 IN IP4 198.51.100.3
...
a=ice-ufrag:3b8fda70
a=ice-pwd:435806ec
a=rtcp:52552 IN IP4 198.51.100.3
a=candidate:Rc6336403 1 UDP 16777215 198.51.100.3 57043 typ relay raddr 198.51.100.131 rport 4000
a=candidate:Rc6336403 2 UDP 16777214 198.51.100.3 52552 typ relay raddr 198.51.100.131 rport 4020
a=remote-candidates:1 203.0.113.3 60181 2 203.0.113.3 60253 a=sendrecv
```

Una de las ventajas de ICE es que la negociación se lleva a cabo entre los UA finales. Gracias a ello, durante el proceso de verificación existe la posibilidad de descubrir nuevos candidatos, los denominados peer-reflexive (prflx).

Estos candidatos se pueden encontrar cuando uno y solo uno de los clientes se encuentra detrás de un NAT dado que los descubre el nodo que NO está detrás de NAT cuando éste comprueba que algún candidato o candidatos de tipo server-reflexive o de tipo host publicados por el host que está detrás del NAT no se corresponden con el origen del mensaje Binding Request.

A continuación, para ilustrar este proceso se muestra un ejemplo utilizando un candidato tipo host en el nodo que está detrás de un NAT. Supongamos el escenario ilustrado en la figura 5.33 en el que:

- Claire dispone de direccionamiento público y será el agente llamante.
- Alice se encuentra detrás de un NAT simetrico y será el agente llamado.
- Tanto Claire como Alice implementan ICE y tienen a su disposición servidores TURN.

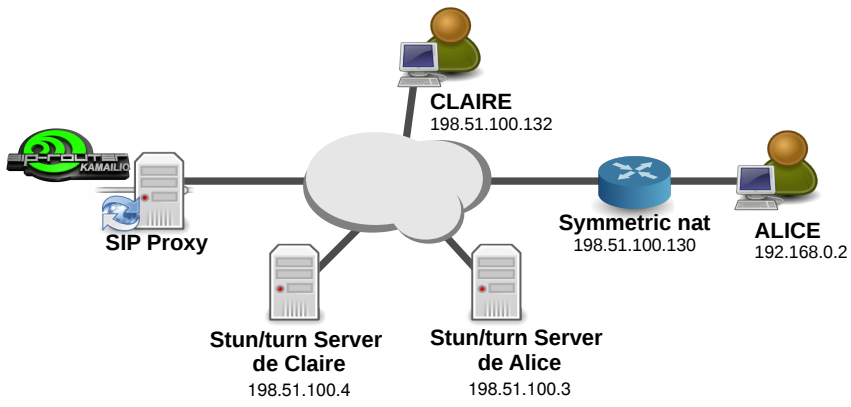


Figura 5.33: ICE - Escenario ejemplo para el descubrimiento de candidatos peer-reflexive.

En este escenario, Claire es el agente llamante y, como hemos visto, envía sus candidatos a Alice en el INVITE:

```
a=candidate:Hc6336484 1 UDP 2130706431 198.51.100.132 4040 typ host
a=candidate:Rcb137569 1 UDP 16777215 203.0.113.3 59862 typ relay
raddr 198.51.100.132 rport 4042
a=candidate:Hc6336484 2 UDP 2130706430 198.51.100.132 4006 typ host
a=candidate:Rcb137569 2 UDP 16777214 203.0.113.3 68269 typ relay
raddr 198.51.100.132 rport 4015


Donde los candidatos involucrados son:
 - Un candidato local para RTP: 198.51.100.132:4040
 - Un candidato local para RTCP: 198.51.100.132:4006
 - Un candidato para RTP a través del servidor TURN: 203.0.113.3:59862
 - Un candidato para RTCP a través del servidor TURN: 203.0.113.3:68269
```

Alice recibe el INVITE y envía sus candidatos a Claire en 200 OK:

```
a=candidate:Hc6336484 1 UDP 2130706431 192.168.0.2 4037 typ host
a=candidate:Rcb007104 1 UDP 16777215 203.0.113.4 62873 typ relay
raddr 198.51.100.130 rport 4002
a=candidate:Hc6336484 2 UDP 2130706430 192.168.0.2 4004 typ host
a=candidate:Rcb007104 2 UDP 16777214 203.0.113.4 62873 typ relay
raddr 198.51.100.130 rport 4006


Donde los candidatos involucrados son:
 - Un candidato local (direccionamiento privado) para RTP: 192.168.0.2:4037
 - Un candidato local (direccionamiento privado) para RTCP: 192.168.0.2:4004
 - Un candidato para RTP a través del servidor TURN: 203.0.113.4:62873
 - Un candidato para RTCP a través del servidor TURN: 203.0.113.4:62873
```

En concreto nos interesa el proceso de validación de los dos pares de candidatos de tipo host ya que los de Alice tienen direccionamiento privado no alcanzable:

- Candidatos RTP: 198.51.100.132:4040 (Claire) <->192.168.0.2:4037 (Alice)
- Candidatos RTCP: 198.51.100.132:4006 (Claire) <->192.168.0.2:4004 (Alice)

Nos centraremos en la validación del par de candidatos para RTP dado que el proceso es idéntico para RTCP. Este proceso de validación, como liustra la figura 5.34, se realiza mediante peticiones STUN de tipo Binding Request de forma que:

- Claire enviará un Binding Request con origen 198.51.100.132:4040 y destino 192.168.0.2:4037 para validar el par de candidatos de RTP.
- Alice enviará un Binding Request con origen 192.168.0.2:4037 y destino 198.51.100.132:4040 para validar el par de candidatos de RTP.

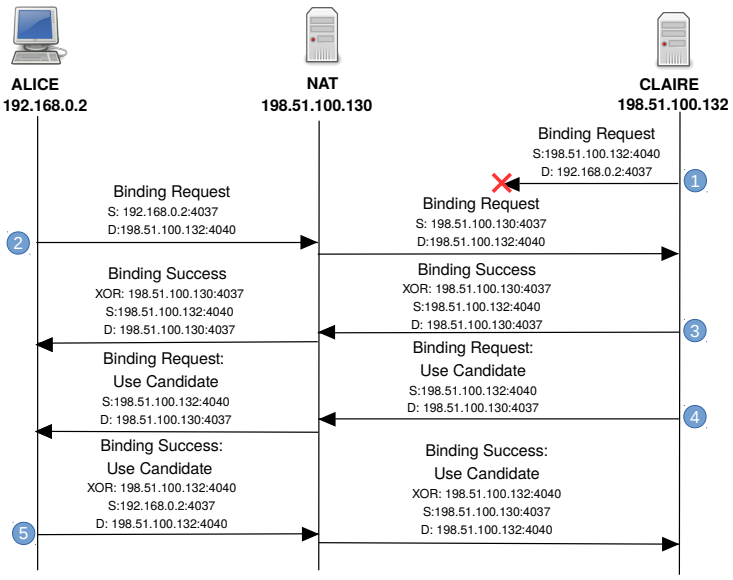


Figura 5.34: ICE - Ejemplo de validación de candidatos.

1. El Binding Request de Claire se pierde dado que el destino es una dirección privada (no alcanzable). Este candidato queda descartado al no recibir respuesta:
2. Alice envía su Binding Request que, en este caso, si que llega a Claire.
3. Claire descubre que el origen de la petición es 198.51.100:130:4037, candidato que Alice no ha informado previamente. Responde al Binding Request de Alice y, dado que la validación se realiza mediante mensajes STUN, incluye además en la respuesta la IP y puertos observados. Gracias a ello, Alice conoce ahora su IP y puerto públicos vistos des del extremo de Claire, esta es una posible dirección peer-reflexive.
4. Claire genera un nuevo "Binding request" pero esta vez al candidato descubierto en el paso anterior para confirmar si es válido y añade además el atributo **USE-CANDIDATE** ya que Claire lo ha nominado para la comunicación RTP dado que este candidato, de tipo peer-reflexive, es preferible al otro candidato de tipo relay. Alice recibe la petición ya que su Binding Request anterior ha abierto un pin hole en el NAT.
5. Alice contesta a Claire con un "Binding Success" informando de la IP y puerto observados que en este caso no varían. Al recibirlo, Claire confirma el nuevo par de candidatos como válidos y serán los elegidos.

Por último, Claire informa a Alice y a todos los servidores intermedios de los candidatos escogidos mediante un UPDATE:

```
UPDATE sip:alice@198.51.100.130:5060;ob SIP/2.0
...
Content-Type: application/sdp
...
o=- 3655999059 3655999060 IN IP4 198.51.100.132
...
m=audio 4040 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 198.51.100.132
a=rtcp:4006 IN IP4 198.51.100.132
...
a=rtcp:4006 IN IP4 198.51.100.132
a=candidate:Hc6336484 1 UDP 2130706431 198.51.100.132 4040 typ host
a=candidate:Hc6336484 2 UDP 2130706430 198.51.100.132 4006 typ host
a=remote-candidates:1 198.51.100.130 4037 2 198.51.100.130 4004
```

Finalmente, Alice actualiza también sus datos de contacto de SDP en el 200 OK del UPDATE:

```
SIP/2.0 200 OK
...
Content-Type: application/sdp
...
o=- 3655999059 3655999060 IN IP4 198.51.100.130
...
m=audio 4037 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 198.51.100.130
a=rtcp:4004 IN IP4 198.51.100.130
...
a=candidate:Pc0a80002 1 UDP 1862270975 198.51.100.130 4037 typ prflx raddr 192.168.0.2 rport 4037
a=candidate:Pc0a80002 2 UDP 1862270974 198.51.100.130 4004 typ prflx raddr 192.168.0.2 rport 4004
```

En resumen:

- Durante el establecimiento de una sesión SIP el candidato original era a través de un relay.
- Durante el proceso de verificación ICE se ha descubierto un nuevo candidato de tipo peer-reflexive address.
- Un candidato peer-reflexive es menos costoso que un candidato relay.
- Por ello, finalmente, el candidato escogido es el descubierto durante el proceso de verificación ICE.

Application Layer Gateway (ALG)

En general, un ALG se define como un elemento de seguridad que aumenta las capacidades de un firewall/NAT hasta la capa de aplicación. En particular, permite implementar filtros de tipo NAT traversal para diversos protocolos de aplicación entre ellos SIP.

Como hemos visto, para que SIP y RTP puedan atravesar un NAT/firewall, los UA deben conocer la combinación dirección y puerto que permite recibir paquetes pero no es la única opción. El ALG, dadas sus capacidades de L7, es capaz de monitorizar el tráfico y abrir los puertos dinámicamente según se precisen.

En el ámbito SIP y en particular en el IMS se utilizan ALGs que dinámicamente abren los puertos y que modifican el SDP/SIP para reflejar estos pin holes. En el ámbito del IMS es usual encontrar al ALG realizando las funciones de firewall/NAT descompuesto en dos entidades:

- Una entidad **(ALG) puramente de control** que es la que maneja las configuraciones.
- Otra entidad **(GW) para la media** que es por la que fluye la media y la que aplica las reglas NAT/firewall.

La entidad de control (ALG) y el gateway (GW) se pueden comunicar mediante el protocolo H.248 quepermite al ALG definir en el GW:

- La asignación y traducción (NAT) de direcciones IP y números de puerto.
- La apertura o cierre de pin holes gracias a la función de filtrado de paquetes en función de IP/puerto.
- Marcar QoS para el tráfico saliente.
- La gestión del RTCP, etc.

Hay que destacar que manipular SIP en un dispositivo no final, como el ALG/GW, rompe el principio "extremo a extremo" y si no se realiza la configuración adecuada, las manipulaciones de un ALG/GW pueden afectar a la señalización SIP y también romper la negociación ICE.

RTP-Proxy

Un proxy RTP es una implementación para NAT traversal en el plano multimedia que funciona independiente del cliente SIP sirviendo como relay para el tráfico RTP en el que el llamante y el llamado envían su tráfico RTP a través de él. Es una solución muy similar a un ALG con media GW en la que el cliente no necesita implementar ningún protocolo.

RTP-Proxy no es un estándar, **es una implementación software** que se suele integrar en soluciones de proxy SIP como Kamailio. La principal diferencia respecto a ALG+GW es que la comunicación entre el proxy SIP y RTP-Proxy se realiza de forma interna con un socket unix, es decir, no utiliza un protocolo estándar como H.248.

Al igual que en el caso anterior, si no se configura adecuadamente, las manipulaciones del RTP-Proxy pueden romper la señalización SIP extremo a extremo o la negociación ICE.

En detalle, RTP-Proxy funciona como sigue:

- Cuando el proxy SIP recibe un INVITE, extrae el `call-id` y se lo envía al RTP-Proxy internamente a través de un socket Unix.
- El RTP-Proxy consulta si ya existe alguna sesión con tal identificación. Si existe dicha sesión devuelve el puerto y protocolo de transporte correspondientes. Si no existe la sesión, el RTP-Proxy crea una nueva sesión y asigna un puerto libre de un rango especificado por configuración y lo informa al proxy SIP.
- El proxy SIP reemplaza la dirección IP y el puerto en el SDP con los obtenidos por el RTP-Proxy y reenvía la petición.
- Cuando proxy SIP recibe una respuesta no negativa con contenido SDP, extrae de nuevo el `call-id` y lo comunica al RTP-proxy que busca la sesión y devuelve el número de puerto asociado. Con esta información, el proxy SIP realiza los cambios correspondientes en el SDP y reenvía la respuesta. En caso de no encontrarla, el RTP-Proxy devuelve un error informando de que no hay ninguna sesión con dicho ID.
- Una vez la sesión SIP ha sido creada, el RTP-proxy escucha en el puerto que ha asignado en espera recibir paquetes de cada una de las dos partes que participan en la llamada.
- Una vez que se reciben estos paquetes, el RTP-proxy rellena una estructura de tipo "ip:puerto" asociada a cada flujo con la ip y puerto origen de los paquetes. Cuando ambas estructuras están completas, es decir, se ha recibido tráfico desde ambas partes, el proxy inicia la transmisión de paquetes entre las ellas.
- El RTP-Proxy controla el tiempo de inactividad para cada una de las sesiones existentes y automáticamente limpia las sesiones cuyos tiempos de inactividad excedan un valor especificado por configuración.

ICE mismatch

En el camino SIP hemos visto que es habitual encontrar dispositivos de NAT traversal como un ALG o un RTP-Proxy. Sin embargo, cuando se utiliza ICE y existen este tipo de dispositivos intermedios se corre el riesgo de romper la negociación.

La modificación del SDP por parte de estos dispositivos hace que no haya ningún candidato ICE que corresponda con la dirección por defecto (atributos *c* y *m* de SDP).

Al recibir el INVITE bajos estas condiciones, el cliente ICE responderá con un atributo *a=ice-mismatch* para indicar que el agente implementa ICE pero este sistema no se puede utilizar debido a la falta de la dirección por defecto en la lista de candidatos.

Supongamos el escenario ilustrado en la figura 5.35:

- El UA "Claire" dispone de direccionamiento público y será el agente llamante mientras que el UA "Alice" se encuentra detrás de un NAT simetrico y será el agente llamado.
- Tanto Claire como Alice implementan ICE y tienen a su disposición servidores TURN en la red pública.

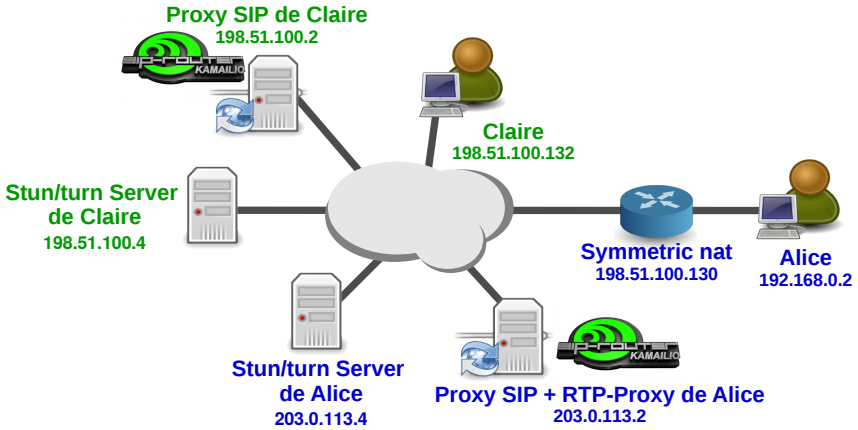


Figura 5.35: Escenario ejemplo para el estudio de ICE mismatch.

Como se ha visto en la sección Interactive Connectivity Establishment (ICE), el INVITE generado por Claire contiene los candidatos tipo host y relay para RTP y RTCP:

```
o=- 3654169836 3654169836 IN IP4 198.51.100.4
...
m=audio 51265 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 198.51.100.4
b=TIAS:64000
a=rtcp:53646 IN IP4 198.51.100.4
...
a=candidate:Hc6336484 1 UDP 2130706431 198.51.100.132 4018 typ host
a=candidate:Rc6336404 1 UDP 16777215 198.51.100.4 51265 typ relay raddr 198.51.100.132 rport 4028
a=candidate:Hc6336484 2 UDP 2130706430 198.51.100.132 4014 typ host
a=candidate:Rc6336404 2 UDP 16777214 198.51.100.4 53646 typ relay raddr 198.51.100.132 rport 4039
```

Sin embargo, al pasar por la combinación proxy SIP + RTP-Proxy de Alice, éste modifica el contenido del SDP al detectar que Alice se encuentra detrás de un NAT forzando que RTP/RTCP se encaminen a través del RTP-Proxy. Así, el INVITE entregado a Alice es:

```
o=- 3654169836 3654169836 IN IP4 203.0.113.2
...
m=audio 56148 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 203.0.113.2
a=rtcp:56149...
a=candidate:Hc6336484 1 UDP 2130706431 \textbf{198.51.100.132 4018} typ host\\
a=candidate:Rc6336404 1 UDP 16777215 \textbf{198.51.100.4 51265} typ relay \\
raddr 198.51.100.132 rport 4028\\
a=candidate:Hc6336484 2 UDP 2130706430 \textbf{198.51.100.132 4014} typ host\\
a=candidate:Rc6336404 2 UDP 16777214 \textbf{198.51.100.4 53646} typ relay \\
raddr 198.51.100.132 rport 4039\\
a=nortpproxy:yes
```

Como se observa, se ha modificado el destino por defecto para RTP/RTCP (campos *m*, *c* y *a* del SDP). Como consecuencia, a pesar de que Alice implementa ICE no lo usará debido a que detecta que no concuerda ningún candidato con el destino por defecto e informa de esta disconformidad o "mismatch" a Claire en el 200 OK:

```
v=0
o=- 3654169836 3654169837 IN IP4 203.0.113.4
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 51690 RTP/AVP 98 96
c=IN IP4 203.0.113.4
b=TIAS:64000
a=rtcp:53802 IN IP4 203.0.113.4
a=sendrecv
a=rtpmap:98 speex/16000
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-16
a=ice-mismatch
```

Para evitar que el RTP-Proxy rompa la negociación ICE es necesario modificar su funcionamiento haciendo que éste añada un nuevo par de candidatos con los recursos reservados para la media:

```
v=0
o=- 3654170736 3654170736 IN IP4 203.0.113.2
m=audio 40486 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 203.0.113.2
b=TIAS:64000
a=rtcp:40487
a=sendrecv
...
a=candidate:Hc6336484 1 UDP 2130706431 198.51.100.132 4031 typ host
a=candidate:Rc6336403 1 UDP 16777215 198.51.100.3 57404 typ relay
raddr 198.51.100.132 rport 4008
a=candidate:Hc6336484 2 UDP 2130706430 198.51.100.132 4032 typ host
a=candidate:20301132 2 UDP 16777214 203.0.113.2 40487 typ relay
a=candidate:20301132 1 UDP 16777215 203.0.113.2 40486 typ relay
a=candidate:Rc6336403 2 UDP 16777214 198.51.100.3 59362 typ relay
raddr 198.51.100.132 rport 4015
a=nortpproxy:yes
```

Universal Plug and Play

UPnP (Universal Plug and Play) es un conjunto de protocolos que permite a equipos de red como ordenadores, impresoras, gateways, APs, etc, descubrir de manera transparente la presencia de otros dispositivos y establecer servicios de red de comunicación.

UPnP implementa una solución para atravesar NAT similar a Application Layer Gateway (ALG) y RTP-Proxy mediante el uso del denominado Internet Gateway Device Protocol (Protocolo IGD).

La idea es que el dispositivo NAT/firewall se expone como Internet Gateway Devices y permite a los nodos UPnP realizar varias acciones. Entre estas acciones tenemos obtener la IP externa del dispositivo, enumerar los mapeos de puertos existentes y añadir o eliminarlos.

Con este protocolo, un UA puede dar instrucciones al dispositivo NAT para abrir un puerto en el lado público y utilizar este puerto para su comunicación.

El inconveniente de UPnP es que está diseñado principalmente para redes domésticas dado que utiliza multicast para la comunicación entre dispositivos. Es decir, solo permite el control de un único dispositivo NAT dentro de cada dominio multicast por lo que no va a funcionar si el UA se conecta en cascada detrás de varios routers.

La otra nota negativa es que la versión actual no cuenta con ningún mecanismo de autenticación lo que supone un serio problema de seguridad.

5.4.4. B2BUA & NAT

Los **Back to Back User Agent (B2BUA)** se pueden utilizar también como una solución al NAT traversal gracias a que son entidades SIP que se encuentran en medio del tráfico SIP y RTP y actúan como agentes de usuario en ambos extremos. Es decir, el B2BUA parte la llamada en dos, generando dos sesiones SIP y dos flujos multimedia.

El beneficio de esta solución es que no requiere ninguna modificación en los clientes y no interfiere con otros protocolos como ICE ya que la negociación ya no es extremo a extremo si no de cada uno de los clientes con el B2BUA. Además, puesto que básicamente se trata de un relay, es capaz de atravesar NAT simétrico con éxito.

Sin embargo, puesto que es un relay, se trata de una solución costosa y hay que tener en cuenta factores como el ancho de banda y la latencia introducida.

5.5. Prácticas

5.5.1. SIP Básico

Escenario

El objetivo de los siguientes ejercicios prácticos es familiarizar al alumno con la señalización básica del protocolo SIP. Para ello, se realizarán diferentes tipos de llamadas utilizando el escenario de la Figura 5.36. Para iniciar el escenario teclee el siguiente comando:

```
hypervisor$ simctl sip-basic start
```

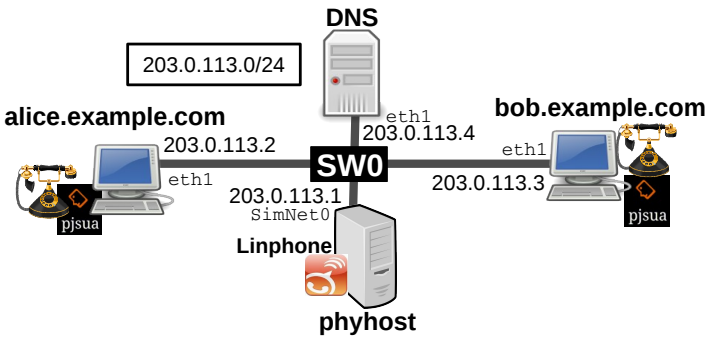


Figura 5.36: Esquema de red para pruebas de SIP básico

Llamada Directa entre Clientes *pjsua*

Ejercicio 5.1– En este ejercicio realizaremos una llamada directa (sin necesidad de intermediarios) entre **alice** y **bob** utilizando el cliente SIP `pjsua`. Capturaremos el tráfico y analizaremos el flujo básico de una sesión SIP.

Note que el usuario llamante debe conocer, como mímimo, la dirección IP o nombre DNS del llamado. Si no se especifica nada más, las aplicaciones SIP utilizan por defecto el protocolo de transporte UDP y el puerto 5060. Una vez aclarado esto, procedemos a abrir una consola de comandos en **alice**:

```
hypervisor$ simctl sip-basic get alice
```

Inicie el cliente *pjsua* en **alice** (que será el llamante) con las siguientes opciones:

```
alice:~# pjsua --null-audio --local-port=5060
```

Notar que en este caso la opción `--local-port` no es estrictamente necesaria ya que el cliente `pjsua` está utilizando el puerto por defecto.

A continuación, en el menú de `pjsua` configure el codec PCMA como prioritario (esto se hace porque nuestro analizador de protocolos `wireshark` no decodifica correctamente speex que es el codificador prioritario de `pjsua`):

```
>>>Cp
Codec name ("*" for all) and priority: PCMA/8000 200
```

A continuación, inicie `pjsua` en **bob** (que será el llamado) con las siguientes opciones:

```
bob:~# pjsua --play-file /root/hello16000.wav --auto-answer 180 --auto-play \
--local-port=5000 --null-audio
```

Dado que **bob** será el usuario llamado y de que no disponemos de un dispositivo de entrada/salida de audio en la máquina virtual, hemos iniciado *pjsua* para que responda de forma automática a una llamada con un tono de ring (Respuesta 180) y para que, al aceptar la llamada, envíe el audio contenido en el fichero /root/hello16000.wav. Además, como puede observar, en **bob** hemos utilizado un puerto local (5000) distinto al puerto por defecto de SIP (5060).

A continuación, iniciamos el analizador de protocolos `wireshark` con el que realizaremos las capturas de tráfico en `SimNet0` (**SW0**) para disponer de todo el tráfico entre las dos máquinas virtuales. Para iniciar `wireshark` en modo captura puede utilizar nuestro script `simtools-captap`. Ejemplo:

```
hypervisor$ simtools-captap -r 0
```

Para poder realizar una llamada desde **alice** a **bob**, en el *pjsua* de **alice** tecleamos:

```
>>> m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:bob@bob.example.com:5000
```

Nota. Para llamar a **bob** puede utilizar su dirección IP (sip:bob@203.0.113.3:5000).

A continuación, aceptamos la llamada en el *pjsua* de **bob** tecleando **"a"** (con **"h"** rechazaríamos la llamada), y respondemos con un 200 OK tecleando 200:

```
 Press a to answer or h to reject call
a
Answer with code (100-699) (empty to cancel): 200
```

Como hemos mencionado, dado que no disponemos de ningún dispositivo de entrada/salida de audio conectador a las máquinas virtuales, no podremos escuchar la llamada, aunque más adelante si la escucharemos gracias a la captura de tráfico. Esperamos unos 10 segundos y finalizamos la llamada tecleando **"h"** en cualquiera de las dos máquinas virtuales. También finalizamos la captura en wireshark: $Capture \rightarrow Stop$

Ahora, vamos a analizar la llamada mediante las herramientas que nos proporciona wireshark en el menú:

$Telephony \rightarrow VoIPcalls$

En esta ventana se puede obtener una visión general de las llamadas capturadas. Los campos presentados son:

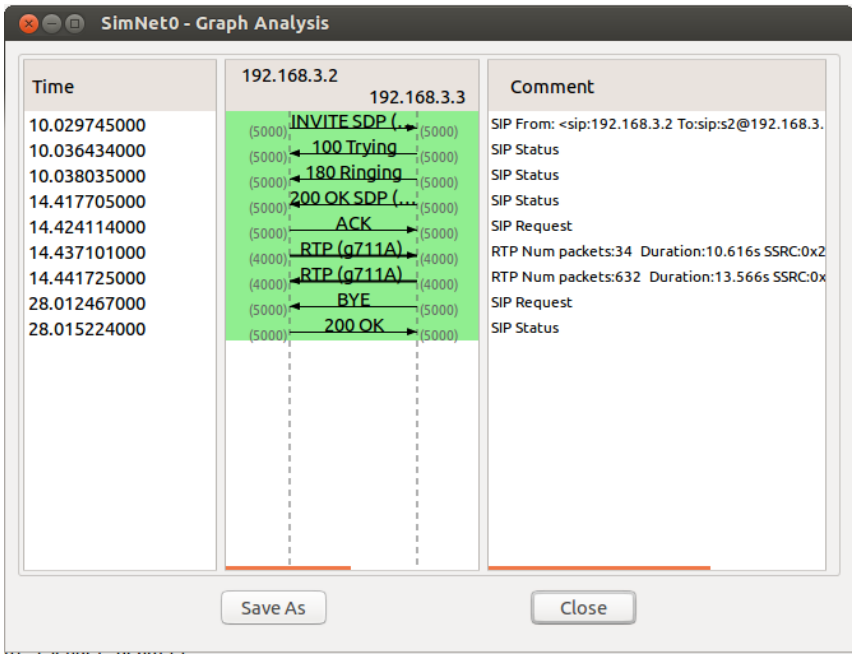
- **Start Time**: Tiempo relativo del inicio de la llamada desde el inicio de la captura.
- **Stop Time**: Tiempo relativo del final de la llamada desde el inicio de la captura.
- **Initial Speaker**: IP del originante de la llamada.
- **From**: Corresponde con la información del campo "From" del INVITE.
- **To**: Corresponde con la información del campo "To" del INVITE.
- **Packets**: Número de paquetes intercambiados durante la llamada.
- **State**: Es el estado en el que se encuentra de la llamada. Los posibles valores son:
  - **CALL SETUP**: Setup, Proceeding, Progress o Alerting
  - **RINGING**: El receptor ha enviado tono de ring
  - **IN CAL**L: Llamada en curso

- **CANCELLED**: El emisor ha cancelado la llamada antes de establecerse.
- **COMPLETED**: La llamada se ha establecido y posteriormente ha finalizado.
- **REJECTED**: El receptor ha cancelado la llamada antes de establecerse.
- **UNKNOWN**: No se puede determinar el estado de la llamada.

El analizador *wireshark* permite visualizar simplificadamente el flujo de un cierta llamada:

$Telephony \rightarrow VoiPcalls \rightarrow Selectall \rightarrow Flow$

Esta vista muestra gráficamente el intercambio de mensajes de la transacción SIP:



En esta vista:

- Pueden aparecer hasta 10 columnas, cada ella representando la dirección IP de los nodos implicados.
- Cada conversación utiliza un color diferente.
- Las flechas indican el sentido de cada paquete.
- La etiqueta encima de cada flecha muestra el tipo de mensaje. Cuando corresponde, también muestra el codec utilizado.
- El tráfico RTP es sumarizado en una flecha con un grosor mayor.
- En cada paquete o sumarización se muestra el puerto TCP/UDP origen y destino.
- La columna de cometarios muestra si el paquete es de tipo “Request” o “Status”.
- El método INVITE muestra además los campos “From” y “To”.

Sin embargo, si queremos analizar en profundidad el intercambio de información, esta vista se queda corta. Vamos a ver otra funcionalidad con la que podremos visualizar todas las cabeceras de la transacción SIP.

Si hacemos click en el paquete correspondiente al método INVITE, éste quedará seleccionado en la captura. Cerramos la visualización gráfica del flujo volviendo a la ventana principal y seleccionamos: $Analyze \rightarrow FollowUDPStream$

Guardamos el diálogo SIP en formato ASCII y también guardamos la captura.

1. ¿Qué métodos aparecen?
2. ¿Qué métodos acepta el emisor? ¿Cómo lo indica?
3. ¿Qué respuestas envía **bob**?
4. ¿A qué responden los diferentes códigos que aparecen? ¿Conoces algún protocolo de la capa de aplicación que use códigos de respuesta similares como por ejemplo [404 - Not Found]?
5. ¿Qué protocolo de la capa de transporte utiliza SIP y RTP? ¿Por qué?

6. ¿Qué puertos utilizan emisor y receptor para el tráfico RTP?
7. ¿Qué codecs de audio propone el emisor? ¿Con cuál de ellos se envía el audio?
8. Además de RTP y SIP, ¿aparecen otros dos protocolos relacionados con VoIP? ¿cuales son? ¿cuál es su propósito?
9. En una comunicación unidirectional ¿quién envía los paquetes RTCP con la información referente al *jitter* y la pérdida de paquetes?

Por último, vamos a escuchar la llamada:

- Iniciamos el menú: $Telephony \rightarrow VoIP$
- Seleccionamos el botón "Player".
- En la nueva ventana seleccionamos el botón "Decode".
- Seleccionamos la llamada y el botón "Play" para escuchar el audio PCMA.

Llamada con Softphone

Ejercicio 5.2– Para la realización de este ejercicio configuraremos un entorno específico para nuestro cliente SIP. En realidad, este "entorno" se llama *namespace* (NS). Este entorno se crea en el **hypervisor** mediante el siguiente comando (debe tener un usuario con privilegios para `sudo` para poder ejecutarlo):

```
hypervisor$ simtools-nsconf -a 203.0.113.1/24 -d 203.0.113.4 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor:~#
```

En el terminal anterior dispondrá de un entorno de red específico (diferente al del resto de aplicaciones del hypervisor). Dispone de una tarjeta de red `eth0` configurada con la IP 203.0.113.1/24 y conectada a la red virtual de nuestro escenario mediante un *switch* con `SimNet0`. También se ha configurado un servidor DNS con dirección 203.0.113.4 y search example.com.

Nota. Puede comprobar la configuración mediante los comandos típicos: `ifconfig`, `route` y `ping`.

A continuación, en el terminal del NS, en el cual es root, iniciaremos el cliente SIP que será un `linphone`:

```
hypervisor$ simtools-nsconf -a 203.0.113.1/24 -d 203.0.113.4 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for youruser:
hypervisor:~# linphone &
```

1. ¿En qué puerto y protocolo escucha por defecto el cliente SIP `linphone`? Puede utilizar un comando `netstat` para averiguarlo en el terminal del NS.

Ahora inicie el cliente `pjsua` en **bob**:

```
bob:~# pjsua --play-file /root/hello16000.wav --auto-answer 200 --auto-play \
--local-port=5000 --null-audio
```

Inicie una captura de tráfico en `SimNet0` y realice una llamada de unos 10 segundos desde el `linphone` a **bob**: $Linphone \rightarrow SIP\,address\,or\,phone\,number :\rightarrow sip : bob@bob.example.com : 5000$

2. ¿Se establece la llamada pudiendo escuchar el audio desde **bob**?
3. ¿Qué datos aparecen en los *headers* Via, From, To, Contact y User-Agent del INVITE enviado por el `linphone`?
4. Observando el SDP de este INVITE, ¿qué puerto utiliza `linphone` para el flujo RTP?

Para acabar el ejercicio cierre el `linphone` y teclee "exit" para salir del NS:

```
hypervisor:~# exit
exit
Deleting NS configuration...
hypervisor$
```

Nota. Asegúrese que la salida del comando anterior se hace de forma correcta, si no es así, avise a un instructor.

Ejercicio 5.3– (*) Vamos a repetir el ejercicio anterior pero esta vez ejecutaremos el `linphone` directamente en el **hypervisor**. Se supone que su **hypervisor** está configurado de forma estándar, es decir, con una tarjeta que tiene una dirección privada y con acceso a Internet. En primer lugar, asignamos 203.0.113.1/24 a `SimNet0` para poder comunicarnos con la red del escenario virtual:

```
hypervisor$ sudo ifconfig SimNet0 203.0.113.1/24
```

Capturando el tráfico en `SimNet0` con `wireshark` conteste a las siguiente preguntas.

1. Para probar la conectividad, mande un `ping` desde el **hypervisor** a **bob**. Utilizando la dirección IP en el `ping` debería obtener respuesta. ¿Puede utilizar el nombre del host **bob** para realizar dicho `ping`? ¿Por qué?

Ejecute el `linphone` desde desde el menú gráfico o bien desde cualquier terminal con su usuario (no privilegiado).

2. ¿Cuál es el valor que muestra el linphone como "*My current identity*"? ¿A qué cree que puede ser debido?

   Pista. Observe la ruta por defecto su **hypervisor**.

3. Ahora vamos a intentar una llamada a **bob**, ¿qué URI utilizará para ello? ¿por qué?
4. Realice una llamada desde el `linphone` a **bob** de unos 10 segundos colgando la misma desde el `linphone`. ¿Se completa la llamada con normalidad?
5. ¿Qué direcciones origen y destino lleva el paquete IP que contiene el mensaje INVITE?
6. Comente el valor de los headers Via, From y Contact del INVITE.
7. Comente el valor del campo c en el SDN del INVITE.

   ```
   c=IN IP4 192.168.181
   ```

   Nuevamente utiliza la IP de la tarjeta donde está definida la ruta por defecto.

8. Comente el valor del header Via de la respuesta 100 Trying enviada por **bob**. ¿A qué dirección IP envía este mensaje SIP **bob**?
9. Respecto al flujo RTP, ¿a qué dirección IP empieza enviando dicho flujo el cliente `pjsua` de **bob**? ¿Y a qué dirección finalmente? Comente también los mensajes de log relacionados que observa en el `pjsua`.

Ejercicio 5.4– (*) En este ejercicio vamos a analizar que sucede cuando cuelga la llamada **bob**. Capturando el tráfico en `SimNet0` con `wireshark` conteste a las siguiente preguntas.

1. Realice una llamada desde el `linphone` a **bob** de unos 10 segundos colgando la misma desde el `linphone`. ¿Se completa la llamada con normalidad?
2. ¿A qué dirección ha enviado el BYE el cliente `pjsua` de **bob**? ¿qué header en la petición INVITE es el que ha hecho que el BYE vaya a esa dirección?

Elimine la ruta por defecto en **bob** (únicamente dejando la ruta directa a 203.0.113.0/24):

```
bob:~# route del default
```

3. Explique qué sucederá cuando se inicia una llamada en el `linphone` hacia **bob** y se cuelga desde el mismo `linphone`.
4. Comente que sucederá con una llamada que se inicia en el `linphone` pero que se cuelga en **bob**.

5.5.2. SIP con Proxies

Escenario

El objetivo de los siguientes ejercicios es iniciar al alumno en escenarios donde hay proxies SIP y en los que se requiere del procedimiento basado en DNS para el descubrimiento y localización de éstos. En primer lugar, iniciaremos en nuestro puesto de trabajo el escenario virtual de la figura 5.37.

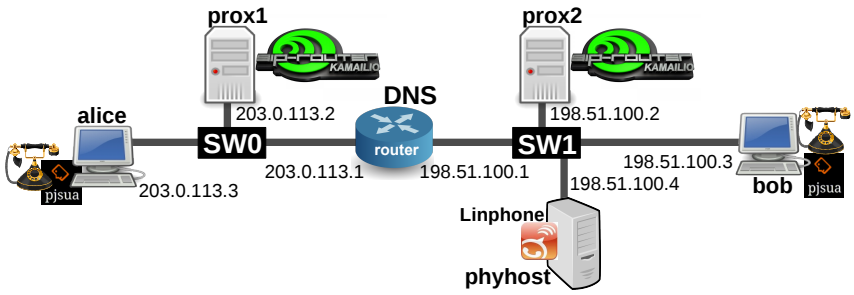


Figura 5.37: Esquema de red

```
hyp$ simctl sip-2proxies start
```

Configuración Inicial

Antes de iniciar los servidores `kamailio`, debemos crear la base de datos que utilizaremos para almacenar toda la información de los usuarios autorizados. Iniciamos un terminal para **prox1** y otro para **prox2** y ejecutamos los siguientes comandos en cada uno de ellos:

```
prox1# kamdbctl create
prox2# kamdbctl create
```

En primer lugar nos pedirá la contraseña del usuario root de mysql. En ambos casos no existe por lo que podemos pulsar la tecla "Enter" e iremos respondiendo "yes" -y- a todas las preguntas que nos haga. A continuación, daremos de alta los usuarios SIP, cada uno en su servidor proxy correspondiente, con el fin de que puedan registrarse y realizar/recibir llamadas:

```
prox1# kamctl add alice xxxx
```

```
prox2# kamctl add bob xxxx
prox2# kamctl add linphone xxxx
```

En ambos casos nos pedirá la contraseña del usuario con permisos de escritura de la base de datos de kamailio. La contraseña es **kamailiorw**. Podemos ver las características de los nuevos usuarios de la siguiente forma:

```
proxX$ kamctl db show subscriber
```

Nótese que en ningún caso hemos especificado el dominio de los usuarios, sin embargo, en las propiedades podemos ver que si tienen un dominio asociado. Eso es debido a que **prox1** se ha configurado para dar servicio al dominio example.org mientras que **prox2** sirve al dominio example.com.

Por último, ya podemos iniciar los servidores `kamailio`:

```
proxX$ service kamailio start
```

Nota. Es recomendable que haga la configuración de forma manual pero en el escenario virtual también dispone de la etiqueta `kamdb` para realizar los pasos anteriores:

```
hyp$ simctl sip-2proxies exec kamdb
```

Para realizar los siguientes ejercicios, configuraremos la traducción de nombres en `wireshark` para que sea más cómodo analizar los diferentes intercambios de mensajes. Para ello, introduzca el siguiente contenido en el fichero `$HOME/.config/wireshark/hosts` (o `$HOME/.wireshark/hosts` dependiendo de donde se almacene su perfil de `wireshark`):

```
203.0.113.1     dnsA
203.0.113.2     prox1
203.0.113.3     alice
198.51.100.1    dnsB
198.51.100.2    prox2
198.51.100.3    bob
198.51.100.4    linphone
```

A continuación, abra un `wireshark` y vaya al menú de `wireshark` Edit->Preferences->Name Resolution y marque las casillas *Only use the profile "hosts" file* y *Resolve network (IP) addresses* tal y como puede observar en la Figura 5.38. Después de realizar los cambios puede cerrar el `wireshark` (ya lo ejecutará cuando sea necesario).

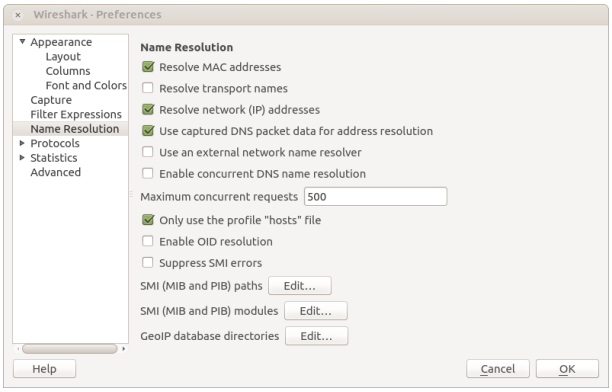


Figura 5.38: Activar un fichero hosts para resolución en Wireshark.

Llamada con un Inbound Proxy

Antes de analizar el escenario trapezoide completo (dos proxies), estudiaremos una configuración más sencilla con un solo proxy. En particular, el objetivo final de los siguientes ejercicios es que un cliente `linphone` en el **hypervisor** pueda completar una llamada a **alice** que dispone de un cliente `pjsua`:

- El cliente en **alice** se registrará en un inbound proxy (**prox1**) para recibir sus llamadas entrantes.
- El cliente `linphone` localizará al proxy de **alice** (**prox1**) utilizando el DNS y los registros NAPTR[2] y enviará la llamada a este proxy. A continuación, el **prox1** enviará la señalización de llamada a **alice**. Finalmente, la comunicación RTP va directa entre el llamado y el llamante, formando un triángulo.

Para la realización de este ejercicio configuraremos un entorno para el cliente `linphone` que vamos a ejecutar desde el **hypervisor**:

```
hypervisor$ simtools-nsconf -t SimNet1 -a 198.51.100.4/25 -d 198.51.100.1 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor:~#
```

En el terminal anterior dispone de un entorno con una tarjeta de red `eth0` configurada con la IP 198.51.100.4/25 y conectada a la red virtual de nuestro escenario mediante un *switch* con `SimNet1`. También se ha configurado 198.51.100.1 como servidor DNS con search example.com.

[2]Notar que los clientes `pjsua` no implementan de forma completa la localización y descubrimiento por DNS (en particular, no soportan registros NAPTR).

El siguiente paso será, desde el terminal del NS, añadir la ruta para la red de **alice**:

```
hypervisor$ simtools-nsconf -t SimNet1 -a 198.51.100.4/25 -d 198.51.100.1 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor:~# route add -net 203.0.113.0/24 gw 198.51.100.1
```

A continuación ejecutamos el `linphone` desde terminal en el NS.

```
hypervisor$ simtools-nsconf -t SimNet1 -a 198.51.100.4/25 -d 198.51.100.1 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor:~# route add -net 203.0.113.0/24 gw 198.51.100.1
hypervisor:~# linphone
```

Ejercicio 5.5– Vamos ahora a analizar el proceso de descubrimiento por DNS para SIP. Para ello, capture tráfico en `SimNet0` y `SimNet1` y realice una llamada desde el `linphone` a **alice**:

$Linphone \rightarrow SIP\,address\,or\,phone\,number :\rightarrow sip : alice@example.org$

Analizando el tráfico en `SimNet0` y `SimNet1` conteste a las siguientes cuestiones:

1. ¿Se establece la llamada?
2. ¿Se observa el INVITE del `linphone` en la captura de `wireshark`?
3. ¿Qué peticiones DNS genera el cliente `linphone`?
4. Según lo visto en la teoría, ¿cuál cree que es la causa de que el cliente `linphone` no llegue ni a generar el INVITE?

Vamos a intentar un registro desde **alice** en su inbound proxy. Para ello, capture tráfico en `SimNet0` y `SimNet1` y ejecute el siguiente comando:

```
alice$ pjsua --null-audio --auto-play --play-file hello16000.wav --auto-answer 200 --add-codec=PCMA/8000 \
--id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice --password xxxx --nameserver=ns1.example.org
```

5. ¿Se realiza el registro?
6. ¿Se observa el REGISTER del `pjsua` en la captura de `wireshark`?
7. ¿Qué peticiones DNS genera el cliente `pjsua`?
8. Según lo visto en la teoría, ¿cuál cree que es la causa de que el cliente `linphone` no llegue ni a generar el REGISTER?

Configuración DNS

En este ejercicio repetiremos la llamada anterior pero antes vamos a configurar en el servidor DNS los registros necesarios para poder localizar a un servidor proxy SIP. Para ello, abrimos una consola en el equipo **router** que es donde reside el servidor DNS:

```
hyp$ simctl sip-2proxies get router
```

Actualizamos el servidor DNS con los siguientes registros:

```
router$ nsupdate -k /etc/bind/rndc.key
> update add example.org. 172800   IN   NAPTR  30  0  "s"   "SIP+D2U"   ""  _sip._udp.example.org.
> update add example.org. 172800   IN   NAPTR  20  0  "s"   "SIP+D2T"   ""  _sip._tcp.example.org.
> update add _sip._udp.example.org. 3600 IN SRV 0 0 5060 prox1.example.org.
> update add _sip._tcp.example.org. 3600 IN SRV 0 0 5060 prox1.example.org.
> send
> update add example.com. 172800  IN   NAPTR    30 0 "s" "SIP+D2U" "" _sip._udp.example.com.
> update add example.com. 172800  IN   NAPTR    20 0 "s" "SIP+D2T" "" _sip._tcp.example.com.
> update add _sip._udp.example.com.   3600    IN      SRV     0 0 5060 prox2.example.com.
> update add _sip._tcp.example.com.   3600    IN      SRV     0 0 5060 prox2.example.com.
> send
```

Nota. Es recomendable que haga la configuración de forma manual pero en el escenario virtual también dispone de la etiqueta `dnsconf` para realizar los pasos anteriores.

Comprobamos que todas las máquinas son capaces de resolver los nuevos registros:

```
$ dig NAPTR example.org
$ dig SRV _sip._udp.example.org
$ dig prox1.example.org
$ dig NAPTR example.com
$ dig SRV _sip._udp.example.com
$ dig prox2.example.com
```

Ejercicio 5.6– Volvemos a repetir la llamada desde el **hypervisor** (`linphone`) a **alice** capturando de nuevo el tráfico en `SimNet0` y `SimNet1`. Conteste a las siguientes cuestiones:

1. ¿Qué protocolo para el transporte de la señalización SIP es el elegido por parte del `linphone`? ¿Es coherente con la teoría del descubrimiento por DNS?
2. ¿Por qué nodos de la red virtual pasa la llamada? ¿por qué no se puede completar esta llamada?

Ejercicio 5.7– Ahora vamos a volver a intentar un registro desde **alice** en su *inbound proxy*. Para ello, capture tráfico en `SimNet0` y ejecute el siguiente comando:

```
alice$ pjsua --null-audio --auto-play --play-file hello16000.wav --auto-answer 200 --add-codec=PCMA/8000 \
--id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice --password xxxx --nameserver=ns1.example.org
```

Analizando el tráfico capturado conteste a las siguientes cuestiones:

1. ¿Por qué tipo de registro SRV pregunta el cliente `pjsua`? ¿qué protocolo de la capa de transporte es el elegido para enviar el REGISTER?
2. Describa el camino SIP que sigue el REGISTER desde que sale de **alice**.
3. Comente el por qué de los valores de la *request line* y de los headers From y To.
4. ¿Cuántas transacciones intervienen en el proceso de registro? ¿cuál es la respuesta generada a cada una de ellas?
5. ¿Cuál es la diferencia entre las dos peticiones de tipo REGISTER? Para ello observe los atributos que se incluyen la cabecera de cada uno de ellos.
6. ¿Por qué cree que el servidor proxy deniega la primera petición de tipo REGISTER?

En **prox1** teclee el siguiente comando:

```
root@prox1:/etc/kamailio# kamctl ul show
```

7. ¿Para qué cree que sirve el comando anterior? Comente los principales parámetros que observa.

Ejercicio 5.8– Capturando de nuevo el tráfico en `SimNet0` y `SimNet1`, volvemos a repetir la llamada desde el **hypervisor** (`linphone`) a **alice** y será esta última la que cuelgue la llamada. Conteste a las siguientes cuestiones:

1. ¿Por qué nodos de la red virtual pasa la señalización de llamada?
2. ¿Por qué aparecen dos mensajes INVITE en `SimNet0`? Comente las diferencias que observa entre ellos, en particular, fíjese en las direcciones IP de ambos así como sus headers Via y Record-Route.
3. Comente el porqué de las diferentes direcciones que aparecen en la request line y en los header Via y Route en cada uno de los dos mensajes ACK capturados en `SimNet0`.
4. Comente el porqué de las diferentes direcciones que aparecen en la request line y en los header Via y Route en cada uno de los dos mensajes BYE capturados en `SimNet0`.
5. Describa las cabeceras Via del 200 OK al BYE. ¿Por qué este mensaje SIP no lleva una cabecera Route?
6. ¿Por qué nodos de la red virtual pasa el flujo RTP?

Llamada con Inbound+Outbound Proxies

En los siguientes ejercicios analizaremos una configuración típica SIP, en la cual, la señalización y el flujo de media forman un trapezoide.

En detalle, la señalización de la llamada entre dos clientes en dos dominios distintos pasará por dos proxies: uno de salida (*outbound*) y otro de entrada (*inbound*). Utilizaremos clientes `pjsua` y el descubrimiento dinámico del *inbound proxy* del otro extremo lo realizará el *outbound proxy* del cliente que realiza la llamada.

Ejercicio 5.9– Con el escenario virtual iniciado y con la configuración correcta (DNS y usuarios), vamos a ejecutar clientes `pjsua` en **alice** y **bob** de forma que se registren en sus respectivos *proxies* con sus usuarios:

```
alice$ pjsua --id sip:alice@example.org  --realm example.org --username alice --password xxxx \
--null-audio --auto-play --play-file /root/hello16000.wav --add-codec=PCMA/8000 \
--auto-answer 180 --nameserver=ns1.example.org \
--registrar sip:example.org --outbound sip:example.org
```

```
bob$ pjsua --id sip:bob@example.com  --realm example.com --username bob --password xxxx \
--null-audio --auto-play --play-file /root/hello16000.wav --add-codec=PCMA/8000 \
--auto-answer 180 --nameserver=ns1.example.com \
--registrar sip:example.com --outbound sip:example.com
```

Observe también que utilizamos la opción `--outbound <proxy_sip>`. Mediante esta opción estamos indicando que todas las llamadas que se realicen desde el cliente utilizaran de forma incondicional al *proxy* (`kamailio`) de su correspondiente dominio. Dicho de otra forma, el *outbound proxy* de cada cliente será el encargado de localizar y encaminar la llamada al destino. Esto permite implementar un escenario SIP en el que el cliente no implementa resoluciones dinámicas DNS-NAPTR (como sucede con `pjsua`).

Una vez registrados los clientes iniciamos capturas de tráfico en `SimNet0` y `SimNet1` y realizamos una llamada de **bob** a **alice**: Analizando el tráfico capturado conteste a las siguientes cuestiones:

1. ¿Qué peticiones DNS realiza el **prox2** para localizar al destino de la llamada?
2. ¿Qué información descubre **prox2** con cada una de las peticiones DNS?
3. ¿Por qué nodos del entorno virtual se encamina la llamada?
4. ¿Cuál es el protocolo de transporte para SIP elegido en cada tramo?
5. ¿En qué escenario cree que sería necesario un outbound proxy a pesar de que el cliente implemente el descubrimiento por DNS?

Ahora capture de nuevo y haga una llamada de **alice** a **bob**.

6. ¿Qué peticiones DNS realiza el **prox1** para localizar al destino de la llamada?
7. ¿Cuál es el protocolo de transporte para SIP elegido en cada tramo? Para dar una explicación consulte el fichero /etc/kamailio/kamailio-local.cfg en el **prox1** y compárelo con ese mismo fichero en **prox2**.

Ejercicio 5.10– En este ejercicio analizaremos un trapezoide pero incluiremos un cliente `linphone`. Para ello configuramos un *namespace* conectado al escenario virtual, añadimos una ruta a 203.0.113.0/24 tal y como se muestra a continuación y finalmente ejecutamos un `linphone` en este namespace:

```
hypervisor$ simtools-nsconf -t SimNet1 -a 198.51.100.4/25 -d 198.51.100.1 -s example.com
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor:~# route add -net 203.0.113.0/24 gw 198.51.100.1
hypervisor:~# linphone
```

Ponga a capturar `SimNet1` y a continuación vamos a registrar al `linphone` en su dominio. Para que el `linphone` se registre en **prox2** utilice el menú:

$Linphone \rightarrow Preferences \rightarrow Manage\ Sip\ Accounts \rightarrow Proxy\ Accounts \rightarrow Add$

Y añada una cuenta con los siguientes datos:

```
Your SIP identity: sip:linphone@example.com
SIP proxy address: <sip:example.com>
Route (optional):
Registration duration (sec): 3600
```

Nos aseguramos que tengamos marcada la opción “Register” y hacemos click en Ok. Nos pedirá usuario y contraseña:

```
user: linphone
password: xxxx
```

1. ¿Qué registros DNS ha utilizado el `linphone` para registrarse?

A continuación registre a **alice**:

```
alice$ pjsua --id sip:alice@example.org  --realm example.org --username alice --password xxxx \
--null-audio --auto-play --play-file /root/hello16000.wav --add-codec=PCMA/8000 \
--auto-answer 180 --nameserver=ns1.example.org \
--registrar sip:example.org --outbound sip:example.org
```

Vuelva a capturar esta vez `SimNet0` y `SimNet1` y realice una llamada de **alice** al `linphone`.

2. ¿Cuál es el camino SIP del INVITE que envía **alice**?

Vuelva a capturar esta vez `SimNet0` y `SimNet1` y realice una llamada del `linphone` a **alice**.

3. ¿Cuál es el camino SIP del INVITE que envía el `linphone`?
4. En el `linphone`, modifique el parámetro que considere necesario en la definición de su cuenta SIP para que la llamada a **alice** pase por todos los nodos de trapecio. Comente que configuración ha utilizado.

ENUM

Ahora vamos a analizar el proceso de descubrimiento ENUM. Para ello, analice el tráfico y realice una llamada desde **bob** al número *tel:10*

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0         For current dialog.
  -1         All 0 buddies in buddy list
  [1 - 0]    Select from buddy list
  URL        An URL
  <Enter>    Empty input (or 'q') to cancel
Make call: tel:10
```

Ejercicio 5.11– Conteste a las siguientes cuestiones:

1. ¿Qué peticiones DNS realiza el **prox2** para localizar al destino de la llamada?
2. ¿Qué información descubre **prox2** con cada una de las peticiones DNS?
3. ¿Qué código de error devuelve **prox2**? ¿Cuál es el motivo?

Como hemos visto en la llamada anterior es necesario configurar en DNS los registros necesarios para traducir una dirección ENUM a una IP y un puerto.

Abrimos la consola del router y configuramos los siguientes registros DNS:

```
router$ nsupdate -k /etc/bind/rndc.key
> update add 0.1.e164.arpa. 172800 IN NAPTR 100 10 "u" "E2U+sip" "!^.*$!sip:alice@example.org!" .
> send
> update add 1.1.nrenum.net. 172800 IN NAPTR 100 10 "u" "E2U+sip" "!^.*$!sip:linphone@example.com!" .
> send
```

Capturamos tráfico y repetimos de nuevo la llamada de **bob** a *tel:10*.

Ejercicio 5.12– Conteste a las siguientes cuestiones:

1. ¿Qué peticiones DNS realiza el **prox2** para localizar al destino de la llamada?
2. ¿Qué información descubre **prox2** con cada una de ellas?
3. ¿Quién recibe la llamada?

Enum E.164 no es el único "listín" público disponible. En el ejercicio anterior hemos configurado en DNS además del que necesitábamos para llamar al destino *tel:10* el registro correspondiente 1.1.nrenum.net.

Utilizar varios listines es tan sencillo como configurar el proxy para que pruebe recursivamente todos los que queramos.

Realizamos una llamada de bob al número *tel:11*:

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: tel:11
```

Ejercicio 5.13– Conteste a las siguientes cuestiones:

1. ¿Qué peticiones DNS realiza el **prox2** para localizar al destino de la llamada? ¿Qué diferencias se observan con el ejercicio anterior?
2. ¿Qué información descubre **prox2** con cada una de las peticiones DNS?
3. ¿Quién recibe la llamada?

Para finalizar la práctica cierre la aplicación `linphone`, haga *exit* en el terminal del *namespace* y pare el escenario de simulación.

5.5.3. SIP y RTP a través de NAT

Escenario

En esta práctica veremos que los problemas derivados de la intervención de NATs en la comuniciación SIP y RTP.

Al finalizarlo los estudiantes serán capaces de:

- Comprender que efecto produce NAT en la comunicación VoIP
- Implementar soluciones NAT-Traversal.

Herramientas para el desarrollo de la práctica

- Tres clientes pjsua
- Dos servidores proxy kamailio
- Un servidor DNS
- Dos servidores STUN
- Dos servidores TURN
- Dos routers NAT
- Wireshark

Configuración del escenario

En primer lugar iniciaremos en nuestro puesto de trabajo el escenario virtual de la figura 5.39:

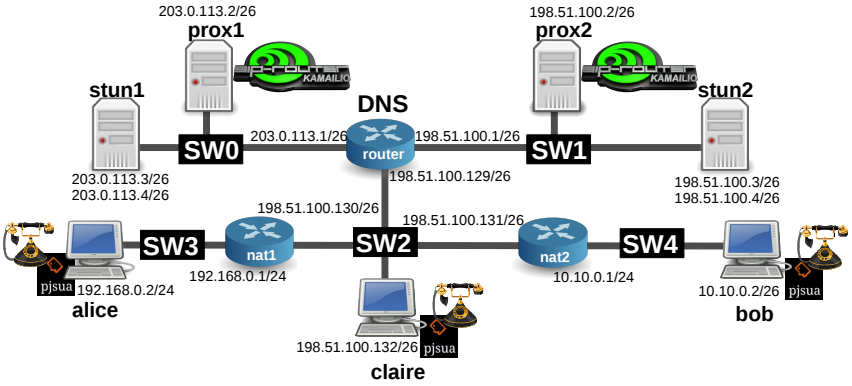


Figura 5.39: Esquema de red

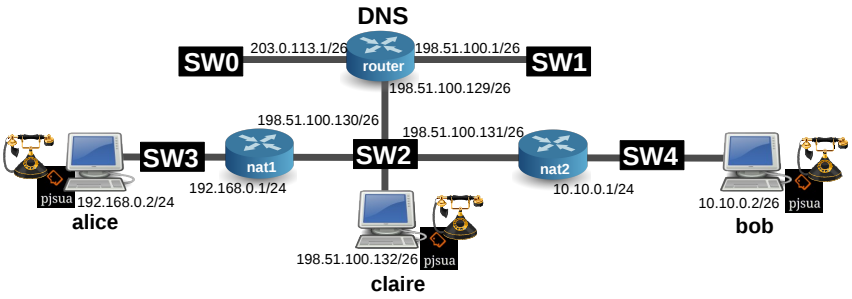


Figura 5.40: Esquema de red

```
phyhost$ simctl sip-nat start
```

En la anterior práctica ya vimos como crear la base de datos con la que trabajará kamailio y añadir usuarios por lo que en esta hemos creado una etiqueta que relizará estos pasos y creará el usuario alice en prox1 y los usuarios bob, claire y linphone en prox2:

```
phyhost$ simctl sip-nat exec kamdb
```

Escenario 1: Llamadas directas

En este escenario analizaremos el comprotamiento de diferentes clientes SIP en llamadas directas cuando en medio de la comunicación se encuentra un elemento de NAT.

El escenario sobre el que iremos desarrollando los ejercicios se muestra en la figura 5.40:

Ejercicio 5.14– En este primer ejercicio del escenario realizaremos una llamada directa entre los clientes claire y alice mediante el softphone pjsua.

Iniciamos los clientes sin registro en sus respectivos servidores kamailio:

```
alice$ pjsua --id sip:alice@example.org  --null-audio --auto-loop \
 --add-codec=PCMA/8000
```

```
claire$ pjsua --id sip:claire@example.com --null-audio \
--add-codec=PCMA/8000 --auto-play --play-file hello16000.wav \
--auto-answer 180
```

Capturamos con Wireshark el tráfico de las interficies SimNet3 y SimNet2 y, a continuación, realizamos una llamada de alice a claire.

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:claire@claire.example.com
```

Como siempre, esperaremos unos 10 segundos antes de terminar la llamada para disponer de tráfico RTP. Con el fin de ilustrar uno de los problemas introducidos por la existencia del NAT en el camino de comunicación, terminaremos la llamada en claire.

- ¿Se cursa correctamente la llamada? ¿Es capaz claire de terminar la llamada?
- ¿Que información aparece en el campo de la cabecera SIP `Contact`? ¿Y en el campo `C` del SDP?
- A raíz de la información obtenida, ¿Cual cree que es la causa de que la llamada no pueda ser finalizada?
- Analizando la llamada con wireshark, ¿llega el RTP a ambos clientes?
- ¿Cual cree que es el motivo de que el RTP enviado por Claire si llegue a Alice? Observe atentamente en wireshark el destino de todos los paquetes RTP generados por Claire y los mensajes que aparecen por pantalla en la consola de Claire.

Ejercicio 5.15– En el ejercicio anterior vimos que el cliente pjsua fue lo suficientemente inteligente para salvar el obstáculo del NAT en el flujo RTP aunque no para el diálogo SIP.

En este ejercicio vamos a ver como se comporta cuando los dos clientes se encuentran detrás de un nat.

Iniciamos los clientes alice y bob sin registro en sus respectivos servidores kamailio:

```
alice$ pjsua --id sip:alice@example.org  --null-audio --auto-play \
--play-file hello16000.wav --auto-answer 180 --add-codec=PCMA/8000
```

```
bob$ pjsua --id sip:bob@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000
```

Capturamos con Wireshark el tráfico de las interficies SimNet4, SimNet3 y SimNet2 y, a continuación, realizamos una llamada de bob a alice.

En DNS hemos reistrados los nombres alice.example.org y bob.example.com con sus respectivas IPs públicas por lo que bob puede hacer la llamada a `sip:alice@alice.example.org`

- ¿Recibe Alice el Invite de Bob?
- ¿Por que elementos del escenario se encamina el INVITE? ¿Cual es el que genera el mensaje "Destination unreachable (Port unreachable)?
- Con la información anterior, ¿Cual cree que es la causa de que no llegue el INVITE a Alice?

Escenario 2: SIP proxy

Hemos visto que no es posible realizar una llamada directa a un cliente que esté detrás de un NAT.

Una de las soluciones sería configurar el NAT para que todo el tráfico que llegue al puerto UDP 5060 se envíe a la dirección IP privada de alice.

Esta solución, sin embargo, no es muy práctica ya que deberíamos conocer a priori que protocolo y puerto va a utilizar el cliente SIP y además, cada cliente destrás de un mismo NAT debería utilizar puertos diferentes para poder dirigir correctamente el tráfico a la red inerna.

En este ejercicio vamos a realizar una llamada entre Alice y Bob con la intervención de sus respectivos servidores proxy kamailio y veremos que el proxy SIP por si solo mitiga en parte la problématica introducida por el NAT pero solo para las transacciones SIP.

El escenario sobre el que desarrollaremos el ejercicio se muestra en la figura 5.41:

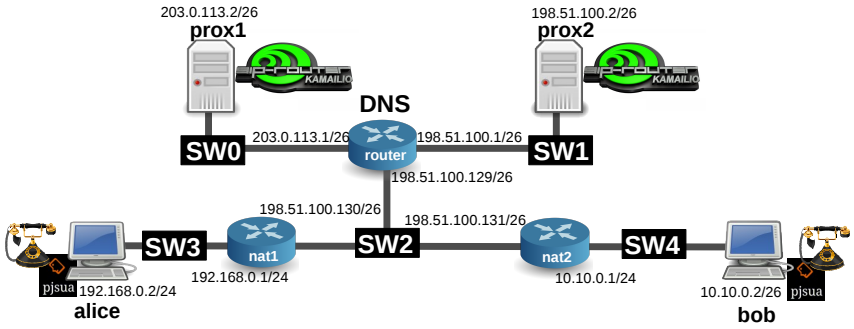


Figura 5.41: Esquema de red

Ejercicio 5.16– Iniciamos los clientes pjsua en bob y alice configurados para registrarse en kamailio.

```
alice$ pjsua --id sip:alice@example.org --null-audio --auto-play \
--play-file hello16000.wav --add-codec=PCMA/8000 --auto-answer 180 \
--realm example.org --username alice --password xxxx  \
--registrar sip:example.org --outbound sip:example.org \
--nameserver=ns1.example.org
```

```
bob$ pjsua --id sip:bob@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username bob --password xxxx --outbound sip:example.com \
--nameserver=ns1.example.com
```

Capturamos con Wireshark el tráfico de las interficies SimNet4, SimNet3 y SimNet2 y realizamos una llamada de bob a alice.

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:alice@example.org
```

Observe que en este caso ya no llamamos a la dirección IP pública de Alice dada de alta en DNS si no que consultamos el servicio SIP del dominio example.org.

Como siempre, esperaremos unos 10 segundos antes de terminar la llamada para disponer de tráfico RTP.

- ¿Se completa correctamente el diálogo SIP?
- ¿Y el tráfico RTP? ¿Llega a su destino?
- ¿Que información aparece en el campo de la cabecera SIP `Contact`? ¿Y en el campo `c - contact information -` del SDP?
- Con la información anterior y observando el contenido del fichero de traducciones del nat en las máquinas Nat1 y Nat2 (*cat /proc/net/ip_conntrack*), ¿cual cree que es la causa de que el tráfico SIP pueda llegar a sus respectivos destinos?

- ¿Cual cree que es la causa que provoca que en este escenario el pjsua no pueda corregir el destino del tráfico RTP?

Escenario 3: STUN

En este escenario 5.42 vamos a realizar una llamada entre Alice y Bob cada uno registrado en su respectivo servidor proxy kamailio pero vamos a configurarlos también para que se apoyen en un servidor STUN y así poder descubrir sus IPs públicas.

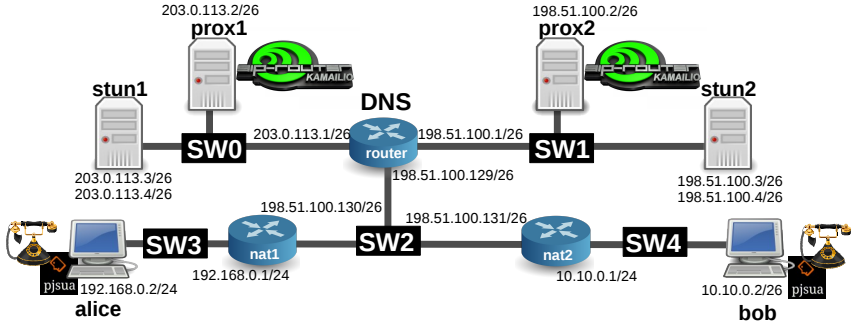


Figura 5.42: Esquema de red

Ejercicio 5.17– Como hemos visto en la teoría, el descubrimiento de los servidores STUN es similar al de los servidores proxy por lo que primero configuraremos en DNS los registros necesarios:

```
router# nsupdate -k /etc/bind/rndc.key
> update add _stun._udp.example.org. 3600 IN SRV 0 0 3478 stun1.example.org.
> update add _stun._udp.example.org. 3600 IN SRV 0 0 3478 stun2.example.org.
> update add _stun._udp.example.org. 3600 IN SRV 0 0 3479 stun1.example.org.
> update add _stun._udp.example.org. 3600 IN SRV 0 0 3479 stun2.example.org.
> update add stun1.example.org 3600 IN  A 203.0.113.3
> update add stun2.example.org 3600 IN A 203.0.113.4
> send
> update add _stun._udp.example.com. 3600 IN SRV 0 0 3478 stun1.example.com.
> update add _stun._udp.example.com. 3600 IN SRV 0 0 3478 stun2.example.com.
> update add _stun._udp.example.com. 3600 IN SRV 0 0 3479 stun1.example.com.
> update add _stun._udp.example.com. 3600 IN SRV 0 0 3479 stun2.example.com.
> update add stun1.example.com. 3600 IN A 198.51.100.3
> update add stun2.example.com. 3600 IN A 198.51.100.4
> send
```

Iniciamos los dos servidores STUN:

```
stunX$turnserver -n -z -S -v
```

```
Los parámetros que estamos configurando son:
-n: Usa solo la configuración indicada parámetros. Es decir, no se usa fichero
    de configuración.
-z: Permite asociaciones anónimas sin autenticación de usuarios.
-v: Verbose
-S: Solo inicia el servidor STUN (sin turn).
```

Iniciamos los clientes pjsua en bob y alice configurados para registrarse en kamailio y utilizar STUN:

```
alice$ pjsua --id sip:alice@example.org --null-audio --auto-play \
--play-file hello16000.wav --add-codec=PCMA/8000 --auto-answer 180 \
--realm example.org --username alice --password xxxx  \
--registrar sip:example.org --outbound sip:example.org \
--stun-srv=example.org --nameserver=ns1.example.org
```

```
bob$ pjsua --id sip:bob@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username bob --password xxxx --outbound sip:example.com \
--stun-srv=example.com  --nameserver=ns1.example.com
```

Capturamos con Wireshark el tráfico de las interficies SimNet4, SimNet3 y SimNet2 y realizamos una llamada de bob a alice:

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:alice@example.org
```

Conteste a las siguientes preguntas antes de finalizar la llamada:

- ¿Se completa correctamente el diálogo SIP?
- ¿Y el tráfico RTP? ¿Llega a su destino?
- ¿Que información aparece en el campo de la cabecera SIP `Contact`? ¿Y en el campo `C` del SDP?
- Con la información anterior y observando el contenido del fichero de traducciones del nat en las máquinas Nat1 y Nat2 (*cat /proc/net/ip_conntrack*), ¿cual cree que es la causa de que el tráfico RTP no pueda llegar a sus respectivos destinos?
- ¿Cual cree que es la causa que provoca que en este escenario el pjsua no pueda corregir el destino del tráfico RTP?

Escenario 4: ICE y TURN

Manteniendo el esquema del escenario anterior, en este ejercicio vamos a realizar una llamada entre Alice y Bob cada uno registrado en su respectivo servidor proxy kamailio y habilitando las funcionalidades TURN e ICE.

Cabe destacar que los clientes pjsua solo utilizan turn si activamos ice por lo que los analizaremos en conjunto.

Ejercicio 5.18– Primero configuraremos en DNS los registros necesarios para resolver el nuevo servicio de TURN:

```
router$ nsupdate -k /etc/bind/rndc.key
> update add _turn._udp.example.org. 3600 IN SRV 0 0 3478 turn1.example.org.
> update add _turn._udp.example.org. 3600 IN SRV 0 0 3478 turn2.example.org.
> update add _turn._udp.example.org. 3600 IN SRV 0 0 3479 turn1.example.org.
> update add _turn._udp.example.org. 3600 IN SRV 0 0 3479 turn2.example.org.
> update add _turn._tcp.example.org. 3600 IN SRV 0 0 3478 turn1.example.org.
> update add _turn._tcp.example.org. 3600 IN SRV 0 0 3478 turn2.example.org.
> update add _turn._tcp.example.org. 3600 IN SRV 0 0 3479 turn1.example.org.
> update add _turn._tcp.example.org. 3600 IN SRV 0 0 3479 turn2.example.org.
> update add turn1.example.org 3600 IN A 203.0.113.3
> update add turn2.example.org 3600 IN A 203.0.113.4
> send
```

```
> update add _turn._udp.example.com. 3600 IN SRV 0 0 3478 turn1.example.com.
> update add _turn._udp.example.com. 3600 IN SRV 0 0 3478 turn2.example.com.
> update add _turn._udp.example.com. 3600 IN SRV 0 0 3479 turn1.example.com.
> update add _turn._udp.example.com. 3600 IN SRV 0 0 3479 turn2.example.com.
> update add _turn._tcp.example.com. 3600 IN SRV 0 0 3478 turn1.example.com.
> update add _turn._tcp.example.com. 3600 IN SRV 0 0 3478 turn2.example.com.
> update add _turn._tcp.example.com. 3600 IN SRV 0 0 3479 turn1.example.com.
> update add _turn._tcp.example.com. 3600 IN SRV 0 0 3479 turn2.example.com.
> update add turn1.example.com. 3600 IN A 198.51.100.3
> update add turn2.example.com. 3600 IN A 198.51.100.4
> send
```

Iniciamos los dos servidores stunX esta vez para que atiendan también peticiones TURN:

```
stunX$turnserver -n -z -v
```

```
Los parámetros són:
-n: usa solo la configuración indicada parámetros
-z: Permite asociaciones anónimas sin autenticación de usuarios.
-v: verbose
```

Iniciamos los clientes pjsua en bob y alice configurados para registrarse en kamailio y utilizar TURN a través de ICE

```
alice$ pjsua --id sip:alice@example.org --null-audio --auto-play \
--play-file hello16000.wav --add-codec=PCMA/8000 --auto-answer 180 \
--realm example.org --username alice --password xxxx  \
--registrar sip:example.org --outbound sip:example.org \
--nameserver=ns1.example.org --use-turn --turn-srv=example.org --use-ice
```

```
bob$ pjsua --id sip:bob@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username bob --password xxxx --outbound sip:example.com \
--nameserver=ns1.example.com --use-turn --turn-srv=example.com --use-ice
```

Capturamos con Wireshark el tráfico de las interficies SimNet4, SimNet3, SimNet2 y SimNet0 y realizamos una llamada de bob a alice esperando unos 10 segundos antes de colgar la llamada.

- ¿Se completa correctamente el diálogo SIP?
- ¿Y el tráfico RTP? ¿Llega a su destino? ¿Por que elementos del escenario se encamina?
- ¿Que información aparece en el campo de la cabecera SIP `Contact`? ¿Y en el campo `C` del SDP?
- ¿En el contenido SDP aparecen un campo nuevo: `a=candidate`. ¿Cual es su función?
- ¿Que tests inicia el cliente pjsua de alice y bob? ¿Cual es el elegido?
- ¿En que captura o capturas somos capaces de escuchar el audio? ¿Que tipo de tráfico sustituye al RTP en las otras?

Ejercicio 5.19– En la teoría hemos visto que hay un tercer candidato que puede descubrirse durante el proceso de verificación de candidatos ICE siempre y cuando uno y solo uno de los clientes se encuentre detrás de un NAT simétrico.

En el ejercicio anterior no se dió esta condición y no vimos ese tipo de candidato. Vamos a analizarlo ahora con una llamada de claire a alice:

```
alice$ pjsua --id sip:alice@example.org --null-audio --auto-play \
--play-file hello16000.wav --add-codec=PCMA/8000 --auto-answer 180 \
--realm example.org --username alice --password xxxx  \
--registrar sip:example.org --outbound sip:example.org \
--nameserver=ns1.example.org --use-turn --turn-srv=example.org --use-ice
```

```
claire$ pjsua --id sip:claire@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username claire --password xxxx --outbound sip:example.com \
--nameserver=ns1.example.com --use-turn --turn-srv=example.com --use-ice
```

Capturamos con Wireshark el tráfico de las interficies SimNet4, SimNet3, SimNet2 y SimNet0 y realizamos la llamada de claire a alice esperando unos 10 segundos antes de colgar la llamada.

- Analizando el método UPDATE y el 200 OK de éste, ¿se ha generado un nuevo candidato ICE diferente a los informados en el INVITE y el 200 OK del INVITE?

Escenario 5: NAThelper y RTPproxy

Como ya hemos visto, la intervenión de NAT en el camino de comunicación de una llamada VoIP representa una grave problemática.

Hemos planteado varias herramientas para solventar esta problemática pero todas ellas requieren que sea el cliente el que se apoye en servicios externos adicionales e implemente los protocolos necesarios para comunicarse con ellos.

Es razonable pensar que lo ideal sería disponer de una solución que fuera independiente del cliente.

Para ello, kamailio dispone de varios módulos de apoyo a NAT que, aunque por si solos no son capaces de presentar una solución, dotan de la inteligencia necesaria al servicio de proxy para detectar cuando nos encontramos ante un cliente conectado a través de una NAT y así poder reencaminar su flujo RTP a través de un relay. Kamailio puede trabajar con diversas soluciones, en este ejercicio, trabajaremos con la solución rtpproxy dado que es la que presenta un mejor rendicimiento a día de hoy.

Bajo este escenario, el cliente es totalmente ajeno a ese proceso de detección y reencaminado y no es necesario que implemente ningún otro protocolo adicional o deba comunicarse con un servicio externo.

Ya tenemos habilitados los módulos de kamailio. Para ello, hemos indiciado en el fichero de configuración de kamailio que trabaje con NAT (/etc/kamailio/kamailio-local.cfg):

```
#!define WITH_USRLOCDB
#!define WITH_NAT
```

Por defecto, kamailio abre un socket de control con rtpproxy a través de la ip de loopback por el puerto 7722 como podemos ver en el fichero de configuración "/etc/kamailio/kamailio.cfg":

```
modparam("rtpproxy", "rtpproxy_sock", "udp:127.0.0.1:7722")
```

Hemos configurado el servicio de rtpproxy para que utilice ese puerto para la comunicación de control y escuche por las direcciones IP públicas respectivas para servir el relay. Por ejemplo en el servidor prox1:

```
prox1# cat /etc/default/rtpproxy
# Defaults for rtpproxy

# The control socket.
#CONTROL_SOCK="unix:/var/run/rtpproxy/rtpproxy.sock"
# To listen on an UDP socket, uncomment this line:
CONTROL_SOCK=udp:localhost:7722

# Additional options that are passed to the daemon.
EXTRA_OPTS="-F -l 203.0.113.2"
```

Ejercicio 5.20– Para completar el escenario sólo nos falta iniciar el servicio en prox1 y prox2:

```
prox1# service rtpproxy start
```

```
prox2# service rtpproxy start
```

Iniciamos la captura en SimNet4, SimNet3 y SimNet2 y volvemos a registrar los clientes pjsua:

```
alice$ pjsua --id sip:alice@example.org --auto-answer 180 --null-audio \
--auto-play --play-file hello16000.wav --add-codec=PCMA/8000 \
--realm example.org --username alice --password xxxx \
--registrar sip:example.org --outbound sip:example.org \
--nameserver=ns1.example.org
```

```
bob$ pjsua --id sip:bob@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username bob --password xxxx --outbound sip:example.com \
--nameserver=ns1.example.com
```

Por último, realizamos una llamada de bob a alice.

- ¿Se completa correctamente el diálogo SIP?
- ¿Y el tráfico RTP? ¿Por que puntos pasa?
- ¿Que información aparece en el campo de la cabecera SIP `Contact`?

  ¿Y en el campo `c` (contact information) del SDP?
- Según la información anterior, ¿A cree que es debido que el tráfico RTP tome ese camino?

Ejercicio 5.21– El uso de rtpproxy es cómodo dado que esta solución es fiable con cualquier tipo de NAT. Sin embargo, un relay siempre es costoso en término de recursos por lo que es interesante mantener la negociación ICE para descubrir una ruta alternativa y más ligera.

Ya hemos visto en la teoría que la modiciación del contenido SDP por parte de una entidad externa (en este caso el proxy) no suele ser buena idea si ésta no es consciente de que se está llevando a cabo una negociación ICE.

En este ejercicio vamos a ilustar ese caso.

Iniciamos los clientes pjsua para negociar ICE ahora que tenemos activado también el RTPproxy:

```
alice$ pjsua --id sip:alice@example.org --auto-answer 180 --null-audio \
--auto-play --play-file hello16000.wav --add-codec=PCMA/8000 \
--realm example.org --username alice --password xxxx \
--registrar sip:example.org --outbound sip:example.org \
--nameserver=ns1.example.org --use-turn --turn-srv=example.org --use-ice
```

```
claire$ pjsua --id sip:claire@example.com --null-audio --auto-loop \
--add-codec=PCMA/8000  --registrar sip:example.com --realm example.com \
--username claire --password xxxx --outbound sip:example.com \
--nameserver=ns1.example.com --use-turn --turn-srv=example.com --use-ice
```

Por último, realizamos una llamada de claire a alice.

- ¿Se completa correctamente el diálogo SIP?
- ¿Y el tráfico RTP? ¿Por que puntos pasa?
- ¿Cual es el destino por defecto indicado en el SDP?
- ¿Con que campo dentro del SDP indica Alice que no va a negociar candidatos mediante ICE?

Ejercicio 5.22– Como hemos visto, el proxy no es consciente de que los clientes van a negociar ICE. Al modificar a ciegas el campo `c` y `m` del SDP, la negociación se rompe y se pierde la posibilidad de encontrar un camino más ligero para RTP.

Para evitarlo, es necesario indicar al proxy que publique la opción del RTPproxy como un candidato más.

Editamos el fichero /etc/kamailio/kamailio.cfg"de *prox1* y *prox2* y añadimos al final la siguiente linea:

```
modparam("rtpproxy", "ice_candidate_priority_avp", "$avp(ice_priority)")
```

Reinicimos en ambos servidores el kamailio para aplicar el cambio:

```
proxX# service kamailio restart
```

Por último, volvemos a realizar una llamada de claire a alice.

- ¿Se completa la negociación ICE?
- ¿Que nuevos candidatos de tipo relay introduce el proxy?
- ¿Que candidatos son los nominados?

5.5.4. Seguridad en SIP y RTP

Escenario

En esta práctica analizaremos SIP y RTP sobre protocolos seguros de la capa de transporte.

Al finalizarlo los estudiantes serán capaces de:

- Implementar SRTP
- Generar autocertificados y claves privadas
- Configurar un proxy SIP para el uso de TLS como protocolo de la capa de transporte

Herramientas para el desarrollo de la práctica

- Tres clientes pjsua
- Dos servidores proxy kamailio
- Un servidor DNS
- Dos servidores STUN
- Dos servidores TURN
- Dos routers NAT
- Wireshark

Configuración del escenario

En primer lugar iniciaremos en nuestro puesto de trabajo el escenario virtual de la figura 5.43:

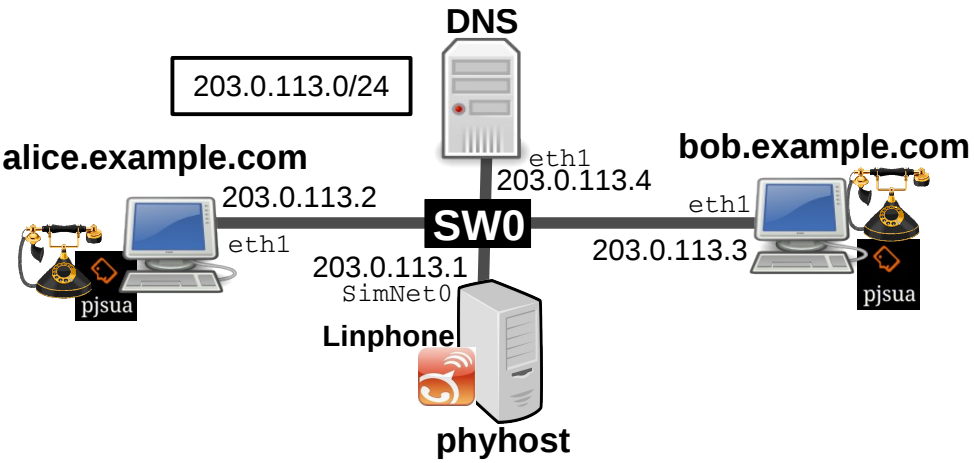


Figura 5.43: Esquema de red

```
phyhost$ simctl sip-srtp-basic start
```

Escenario 1: Llamadas directas

En este escenario vamos a analizar SRTP y SIP sobre TLS en un escenario muy básico en el que sólo intervienen dos clientes pjsua.

Ejercicio 5.23– Realizaremos una llamada directa entre los clientes **bob** y **alice** utilizando SRTP (SDES) y SIP inseguro. Iniciamos los clientes sin registro en sus respectivos servidores kamailio:

```
alice# pjsua --id sip:alice@alice.example.com  --null-audio \
--auto-play --play-file hello16000.wav --auto-answer 180 \
--add-codec=PCMA/8000 --use-srtp=1 --srtp-secure=0
```

```
bob# pjsua --id sip:bob@bob.example.com --null-audio --auto-loop \
--add-codec=PCMA/8000 --use-srtp=1 --srtp-secure=0
```

Hemos introducido dos nuevos parámetros de configuración del cliente pjsua:

```
-use-srtp=N

Donde:

0: SRTP deshabilitado.
1: SRTP activo pero no mandatori. Es decir, el cliente informa de que soporta
SRTP y lo utilizará en caso que el otro extremo también lo soporte. En caso
contrario usará RTP.
2: SRTP mandatori. La llamada solo se establecerá si los dos extremos
negocian SRTP.
```

```
-srtp-secure=N

Donde:

0: SRTP no requiere señalización segura. Es decir, no requiere un protocolo
de transporte seguro como TLS para la señalización.
1: SRTP requiere un protocolo de transporte seguro como TLS para
la señalización.
2: SRTP requiere que la señalización extremo a extremo sea mediante
un protocolo de transporte seguro (URI sips:).
```

Capturamos con Wireshark el tráfico de la interficie SimNet0 y realizamos una llamada de **bob** a **alice**. Como siempre, esperaremos unos 10 segundos antes de terminar la llamada para disponer de tráfico SRTP.

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:alice@alice.example.com
```

Por último, guardamos la captura de wireshark.

- ¿Que atributos nuevos aparecen en el SDP?
- ¿Cual es su función?
- ¿Es posible reproducir el audio correctamente desde la captura de wireshark?

Ejercicio 5.24–

En el ejercicio anterior hemos securizado el canal de la media con SRTP y cifrado SDES. Sin embargo, hemos visto que las claves de cifrado y el método se informa en texto plano dentro de SDP lo que supone un alto riesgo de seguridad ya que nos permite descifrar fácilmente esos datos.

En este ejercicio vamos a ver como descifrar el flujo SRTP protegido por SDES de Alice.

De la captura del ejercicio anterior nos interesan dos datos: la clave de cifrado y los puertos del flujo RTP.

La clave de cifrado que utiliza Alice es la cadena alfanumérica que aparece en el atributo "inline" del parámetro "`a=crypto`" de la respuesta "200 OK" de Alice. Algo similar a lo mostrado en la figura 5.44. Anotamos esta cadena.

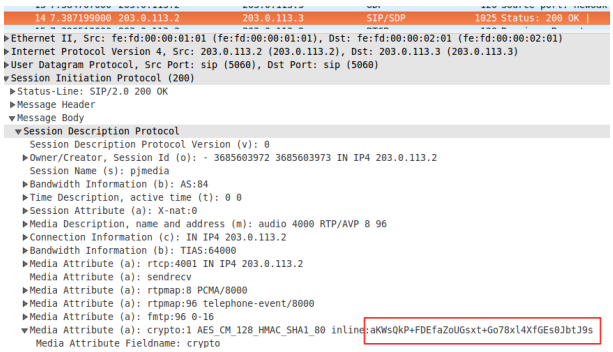


Figura 5.44: Clave de cifrado negociada para SRTP

Para copiarla, nos situamos en la linea en la que aparece, y con el botón derecho seleccionamos la opción $Copy \rightarrow Value$.

Los puertos del flujo RTP, como hemos visto en teoría, podemos encontrarlos en el campo Media description, "m", del "INVITE" y "200 OK" respectivamente. La figura 5.45 ilustra como determinarlos a través de wireshark.

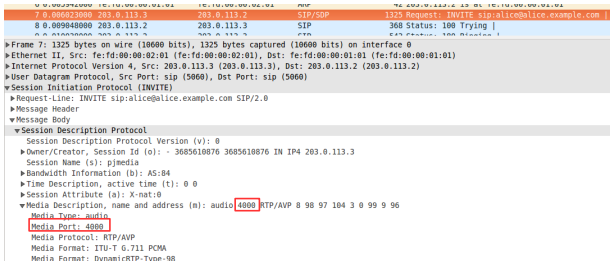


Figura 5.45: Puerto RTP indicado en el SDP del INVITE

Una vez disponemos de la clave y puertos involucrados, filtramos en wireshark el tráfico de forma que sólo veamos los paquetes correspondientes al flujo RTP de Alice a Bob. Para ello escribimos "**ip.src == 203.0.113.2 and udp.port == *<alice_rtp_port>***" en el editor de filtros, donde *<alice_rtp_port>* es el puerto indicado por Alice para el RTP. La figura 5.46 muestra un ejemplo con el puerto 4000.

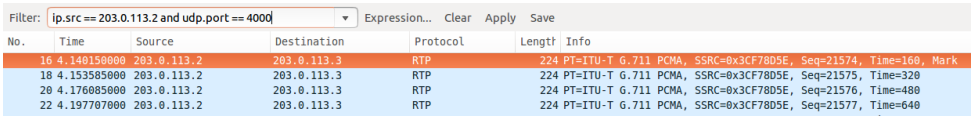


Figura 5.46: Filtro para el tráfico RTP de Alice en wireshark

Exportamos el flujo RTP a través del menú $Edit \rightarrow ExportSpecifiedPackets$. Guardaremos la captura en local con el nombre "*rtp-sdes*" y el formato "Wireshardk/tcpdump/...-pcap" seleccionable en el desplegable $Filetype$

como muestra la figura 5.47. Antes de realizar la exportación, nos aseguraremos también que la opción $Displayed$ es la seleccionada.

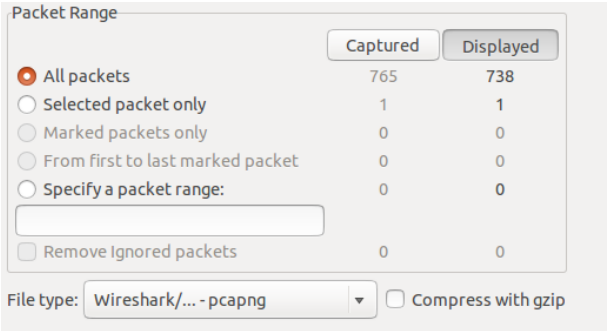


Figura 5.47: Filtro para el tráfico RTP en wireshark

Vamos ahora a descifrar este flujo. Nos situamos en la carpeta "soft/srtp-decrypt-master" que se encuentra en el path de este escenario.

En esta carpeta tenemos una aplicación capaz de descifrar SRTP con cifrado SDES. Para ello, ejecutamos el siguiente comando:

```
$ ./srtp-decrypt -k clave_de_cifrado < rtp-sdes.pcap > rtp-sdes.txt
```

Donde:

- "*clave_de_cifrado*": Es la clave de cifrado que hemos copiado. Es decir, en el ejemplo de la figura 5.44 sería la cadena delimitada por el rectángulo rojo.
- "< rtp-sdes.pcap": Representa el fichero de entrada, es decir, la captura a leer.
- "> rtp-sdes.txt": Representa el fichero de salida. Este contendrá el flujo RTP descifrado en formato hexadecimal.

Ya tenemos el flujo descifrado y, para comprobarlo, lo volveremos a importar en wireshark y lo escucharemos pero, dado que el formato de los datos es hexadecimal, tendremos que seguir los siguientes pasos para que wireshark interprete correctamente el contenido.

Importamos el fichero "rtp-sdes.txt" en wireshark a través de $Edit \rightarrow Import from Hex Dump$ con las opciones que se muestran en la figura 5.48.

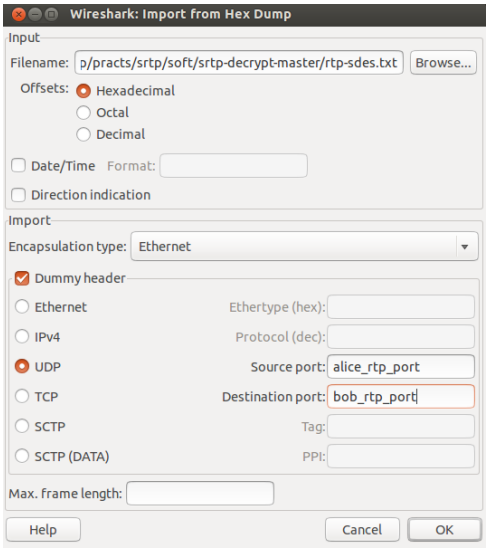


Figura 5.48: Opciones para importar el fichero en hexadecimal con el tráfico RTP

Como muestra la figura 5.49, veremos que se han perdido los datos de IP origen y destino y que el protocolo indicado ya no es RTP si no UDP.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	1.1.1.1	2.2.2.2	UDP	214	Source port: terabase Destination port: terabase
2	0.000001000	1.1.1.1	2.2.2.2	UDP	214	Source port: terabase Destination port: terabase
3	0.000002000	1.1.1.1	2.2.2.2	UDP	214	Source port: terabase Destination port: terabase
4	0.000003000	1.1.1.1	2.2.2.2	UDP	214	Source port: terabase Destination port: terabase

Figura 5.49: Protocolo RTP no reconocido

Mientras que las IPs nos dan igual, si que tendremos que indicar a wireshark que se trata del protocolo RTP para poder escucharlo. Para ello nos situamos encima de cualquier paquete y con el botón derecho seleccionamos $DecodeAs$ y RTP en el listado que nos aparecerá. Una vez aplicado, veremos que el protocolo vuelve a ser RTP como se muestra en la figura

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	1.1.1.1	2.2.2.2	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5DF7EE58, Seq=26184, Time=160, Mark
2	0.000001000	1.1.1.1	2.2.2.2	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5DF7EE58, Seq=26185, Time=320
3	0.000002000	1.1.1.1	2.2.2.2	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5DF7EE58, Seq=26186, Time=480
4	0.000003000	1.1.1.1	2.2.2.2	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5DF7EE58, Seq=26187, Time=640

Figura 5.50: Protocolo RTP reconocido

Por último, escucharemos el audio para confirmar que lo hemos descifrado correctamente. Esta vez, dado que sólo tenemos RTP, a través del $Player$ dentro de $Telephony \rightarrow RTP \rightarrow StreamAnalysis$ con los checkbox seleccionados que se muestran en la figura 5.51.

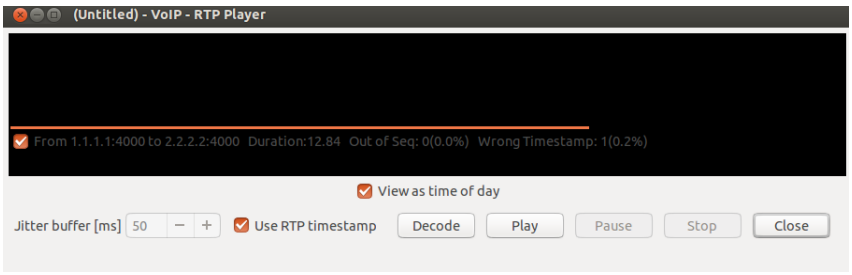


Figura 5.51: Reproducción del audio descifrado

Ejercicio 5.25– En este ejercicio vamos a dar un paso más en seguridad y utilizar SIP sobre TLS de forma que la señalización también vaya cifrada.

Iniciamos los clientes **alice** y **bob** pero esta vez indicando que queremos usar TLS.

```
alice# pjsua --id sip:alice@alice.example.com  --null-audio \
--auto-play --play-file hello16000.wav --auto-answer 180 \
--add-codec=PCMA/8000 --use-srtp=1 --use-tls
```

```
bob# pjsua --id sip:bob@bob.example.com --null-audio \
--auto-loop --add-codec=PCMA/8000 --use-srtp=1 --use-tls
```

Realizamos una llamada de bob a alice indicando que ha de ser sobre transporte tls:

```
sip:alice@alice.example.com;transport=tls
```

- ¿Se establece la llamada?
- ¿Que mensajes de error aparecen en las consolas de **bob** y **alice**?
- Analizando los mensajes y la captura de wireshak, ¿Cual cree que es la causa?

Ejercicio 5.26– Para poder completar el handshake de TLS es necesario que el servidor disponda de, al menos, un par certificado público y clave privada.

En este caso el cliente que hace las veces de servidor es **alice** ya que es quién recibe la llamada.

Vamos a crear en **alice** unos certificados autofirmados para poder usar SIP sobre TLS.

1. En primer lugar, generamos una clave privada aleatoria de 1024bits, cifrada con triple-des y protegida con la contraseña que nos pedirá al ejecutar el siguiente comando:

```
alice$ openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
........++++++
.........................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

   Es indiferente la contraseña que especifique, más adelante la eliminaremos. Use por ejemplo "**1234**".

2. Generamos la petición de certificado. Nos pedirá rellenar una serie de campos, podemos dejar todos sin modificar:

```
alice$ openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Eliminamos la contraseña de la clave privada:

```
alice$ cp server.key server.key.org
alice$ openssl rsa -in server.key.org -out server.key
Enter pass phrase for server.key.org:
writing RSA key
```

4. Por último, generamos un certificado público y eliminamos los ficheros que ya no son necesarios:

```
alice$ openssl x509 -req -days 365 -in server.csr -signkey server.key \
-out server.crt
alice$ rm server.csr server.key.org
```

Ya podemos iniciar el cliente pjsua para que utilice los certificados generados:

```
alice# pjsua --id sip:alice@alice.example.com  --null-audio --auto-play \
--play-file hello16000.wav --auto-answer 180 --add-codec=PCMA/8000 \
--use-srtp=1 --use-tls --tls-cipher 0x000035 \
-tls-cert-file server.crt -tls-privkey-file server.key
```

```
bob# pjsua --id sip:bob@bob.example.com --null-audio \
--auto-loop --add-codec=PCMA/8000 --use-srtp=1 --use-tls \
--tls-cipher 0x000035
```

Iniciamos la captura en wireshark y volvemos a relaizar la llamada de **bob** a **alice** usando TLS como protocolo de la capa de transporte como vimos en el ejercicio anterior:

- ¿Se establece la llamada?
- ¿Somos capaces de analizarla con wireshark? ¿Detecta éste que se haya realizado alguna llamada?

Dado que SIP ahora se encapsula sobre TLS, toda la información viaja cifrada y por lo tanto no somos capaces de analizarla.

Wireshark, sin embargo, permite descifrar SSL siempre que se disponga de la clave privada y los cifrados utilizados no utilizen DH (Diffie-Hellman).

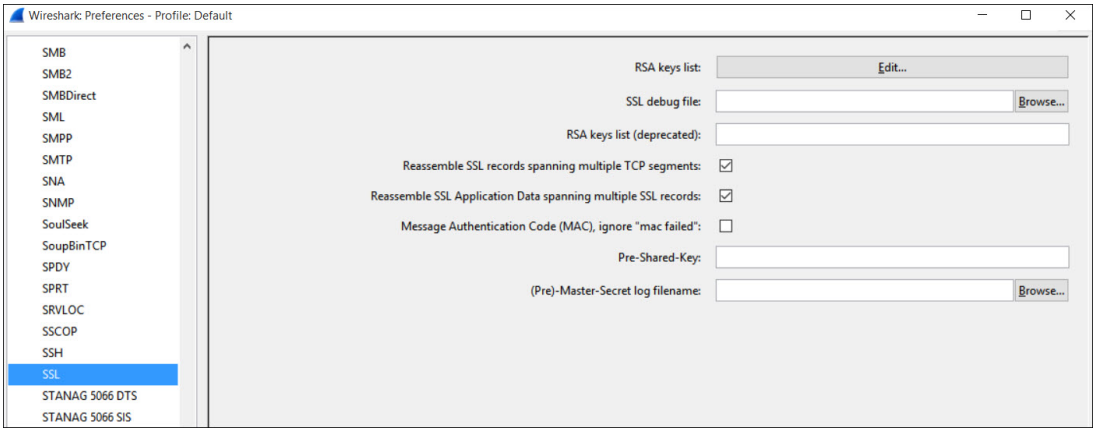
Es por ello que hemos iniciado los clientes pjsua para que utilicen un cifrado que wireshark sea capaz de descifrar:

```
--tls-cipher 0x000035

Donde el código 0x000035 es el que identifica al cifrado AES256-SHA
```

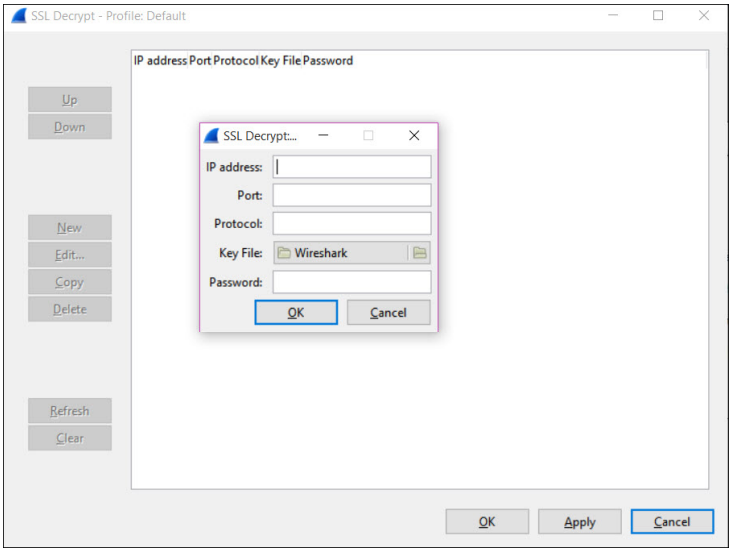
Vamos a descifrar el tráfico capturado:

1. Copiamos en el host local la clave privada de alice. La guardamos en el fichero **/key-alice.key**
2. En wireshark seleccionamos: $Edit \rightarrow Preferences \rightarrow Protocols \rightarrow SSL$



3. Hacemos clic en el botón edit y añadimos la clave de alice con los siguientes parámetros:

```
IP address: 203.0.113.2
Port: 5061
Porotocol: sip
Key file: /key-alice.key
```



4. Aplicamos y aceptamos hasta salir de la ventana de preferencias.

- ¿Somos capaces ahora de analizar el tráfico SIP? Fíjese que ahora para seguir la conversación es necesario seleccionar "*Follow SSL Stream*"

Escenario 2: SIP proxy

Ahora que ya tenemos una visión básica de seguridad, vamos a analizar y configurar un escenario más complejo incorporando Proxy SIP.

En primer lugar iniciaremos en nuestro puesto de trabajo el escenario virtual de la figura 5.52:

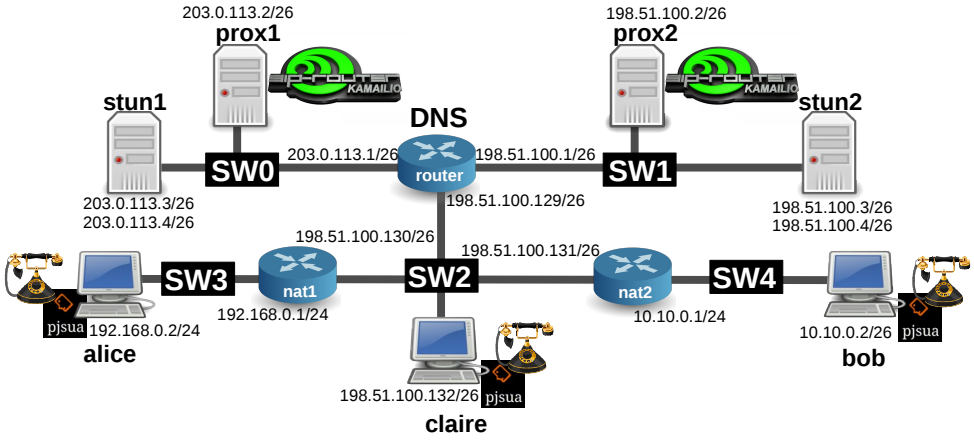


Figura 5.52: Esquema de red

```
phyhost$ simctl sip-srtp start
```

Ejercicio 5.27– En este escenario los servidores proxy son lo que deben disponer de certificado y clave privada por volveremos a ejecutar en **prox1** y **prox2** el procedimiento que hemos visto para generarlos.

Una vez generados, copiaremos los ficheros al directorio */etc/kamailio/* y configuramos los dos proxy para trabajar sobre TLS.

En el fichero */etc/kamailio/kamailio-local.cfg* habilitamos TLS:

```
#!define WITH_TLS
```

En el fichero */etc/kamailio/tls.cfg* configuramos los parámetros relativos a los certificados:

```
[server:default]
method = TLSv1
verify_certificate = no
require_certificate = no
private_key = /etc/kamailio/server.key
certificate = /etc/kamailio/server.crt
...
[client:default]
verify_certificate = no
require_certificate = no
```

Reiniciamos el servicio kamailio para aplicar los cambios:

```
proxX# service kamailio restart
```

Fíjese que hemos modificado el parámetro *verify_certificate*. Esto es debido a que los certificados que vamos a usar son autofirmados y por tanto no confiables. Si no evitamos verificar el certificado, los kamailio rechazarán la comunicación SIP entre ellos.

A continuación configuramos en DNS los registros necesarios para resolver el servico sobre TLS:

```
router# nsupdate -k /etc/bind/rndc.key
> update add example.org. 3600  IN      NAPTR   40 0 "s" "SIPS+D2T" "" \
_sips._tcp.example.org.
> update add _sips._tcp.example.org.      3600    IN      SRV     0 0 \
5061 prox1.example.org.
> send
> update add example.com. 3600  IN      NAPTR   40 0 "s" "SIPS+D2T" "" \
_sips._tcp.example.com.
> update add _sips._tcp.example.com.      3600    IN      SRV     0 0 \
5061 prox2.example.com.
> send
```

Iniciamos la captura con wireshark en SimNet2. Es importante iniciar la captura de forma previa al registro de los clientes ya que necesitamos capturar el handshake de ssl para poder descifrarlo.

Configuramos los clientes para que utilicen TLS:

```
alice# pjsua --id sips:alice@example.org  --registrar sips:example.org \
--realm example.org --username alice --password xxxx --null-audio \
--auto-play --auto-answer 180 --play-file hello16000.wav \
--outbound "sips:example.org" --nameserver=ns1.example.org \
--add-codec=PCMA/8000 --use-tls --use-srtp=1 --srtp-secure=21 \
--tls-cipher 0x000035
```

```
bob# pjsua --id sips:bob@example.com  --registrar sips:example.com \
--realm example.com --username bob --password xxxx --null-audio \
--auto-loop --outbound "sips:example.com" --nameserver=ns1.example.com  \
--add-codec=PCMA/8000 --use-tls --use-srtp=1 --srtp-secure=1 \
--tls-cipher 0x000035
```

Realizamos una llamada de **bob** a **alice** y analizamos por que elementos del escenario se encamina la senalización y por que elementos la media.

```
sip:alice@example.org
```

1. ¿Por que elementos del escenario se encamina el tráfico SIP? ¿Y el tráfico RTP?
2. ¿Que protocolo de la capa de transporte se utiliza en cada tramo? Fíjese por ejemplo en el utilizado entre los dos proxy SIP.
3. ¿A que cree que se debe ese cambio de protocolo? Recuerde que método utiliza un Proxy SIP para descubrir el destino de una llamada.

Ejercicio 5.28–

Acabamos de ver que a pesar de securizar la conexión entre nuestro cliente SIP y su Proxy, es posible que la señalización no viaje sobre un protocolo seguro extremo a extremo.

Para ello, necesitaremos usar un esquema de URI que nos asegure que la información viajará cifrada des de el inicio hasta el final.

En este ejercicio volveremos a capturar una llamada de **bob** a **alice** pero utilizando el esquema SIPS:

```
sips:alice@example.org
```

1. ¿Que protocolo de la capa de transporte se utiliza ahora en cada tramo?
2. ¿A que se debe que en esta ocasión los Proxy no obedezcan al descubrimiento por DNS?

Capítulo 6

IMS

6.1. Introducción a IMS

Históricamente, la telefonía han proporcionado servicios de llamadas de voz a través de una red de conmutación de circuitos. La implementación, despliegue y evolución de servicios en redes, tanto fijas como móviles, basadas en circuitos es una labor tediosa mientras que estas mismas operaciones en las redes de datos IP se hacen de forma mas ágil y flexible. Si bien existen desde hace años implementaciones de servicios multimedia sobre IP, como por ejemplo Skype, el servicio prestado por los operadores hasta hace poco se seguía prestando sobre redes de circuitos.

IMS (IP Multimedia Subsystem) es un estándar internacional diseñado originariamente por la 3GGP (Third Generation Partnership Project) en colaboración con el IETF (Internet Engineering Task Force) que ha sido adoptado también por otros organismos de estandarización como 3GPP2 y ETSI. La especificación IMS estandarizada por 3GPP mediante la colaboración con IETF ha facilitado la integración con Internet siempre que sea posible (por ejemplo gracias al uso del protocolo SIP).

Se trata de un marco arquitectónico basado en los conceptos de NGN (Next Generation Networks) capaz de proveer sesiones multimedia en tiempo real, como sesiones de voz, video y conferencia, y sesiones que no requieren de tiempo real, como presencia o mensajería instantánea, sobre infraestructura IP. Pretende trasladar la facilidad en la concepción de servicios del mundo de Internet al de las redes de telefonía.

Además, el estándar es independiente de la tecnología de acceso (GSM, GPRS, UMTS, HSDPA, DSL, HFC, Wi-Fi, Wi-Max, Bluetooth, etc.) lo que aporta ventajas como pasar de un sistema a otro sin interrumpir la conexión o el acceso simultáneo desde varios medios. IMS permite a los operadores controlar la calidad de servicio de cada aplicación de extremo a extremo y proporciona acceso y autenticación de forma segura basada en diferentes métodos como credenciales de acceso o una tarjeta SIM. También permite la incorporación de nuevos servicios como "presencia" para ver quién está disponible en un momento dado o mensajería instantánea para el intercambio de información en tiempo real.

Para el usuario final ofrece nuevas opciones de comunicación que combinan sesiones de voz con elementos multimedia como vídeo o juegos en línea con otros usuarios de la red mientras hay una llamada en curso dando cabida a nuevas formas de comunicación.

6.2. Arquitectura IMS

Antes de explorar la arquitectura general en el IMS hay que señalar que el 3GPP no habla de nodos si no de funciones. Esto significa que la arquitectura IMS es una colección de funciones vinculadas por interfaces estandarizadas.

Las especificaciones de IMS definen funciones para controlar la señalización y el tráfico de suscriptor para aplicaciones multimedia. Se compone de un diseño modular separado en cuatro capas lógicas: la capa de servicio, la capa de control, la capa de transporte y la capa de acceso. La Figura 6.1 muestra una vista de alto nivel de la arquitectura IMS.

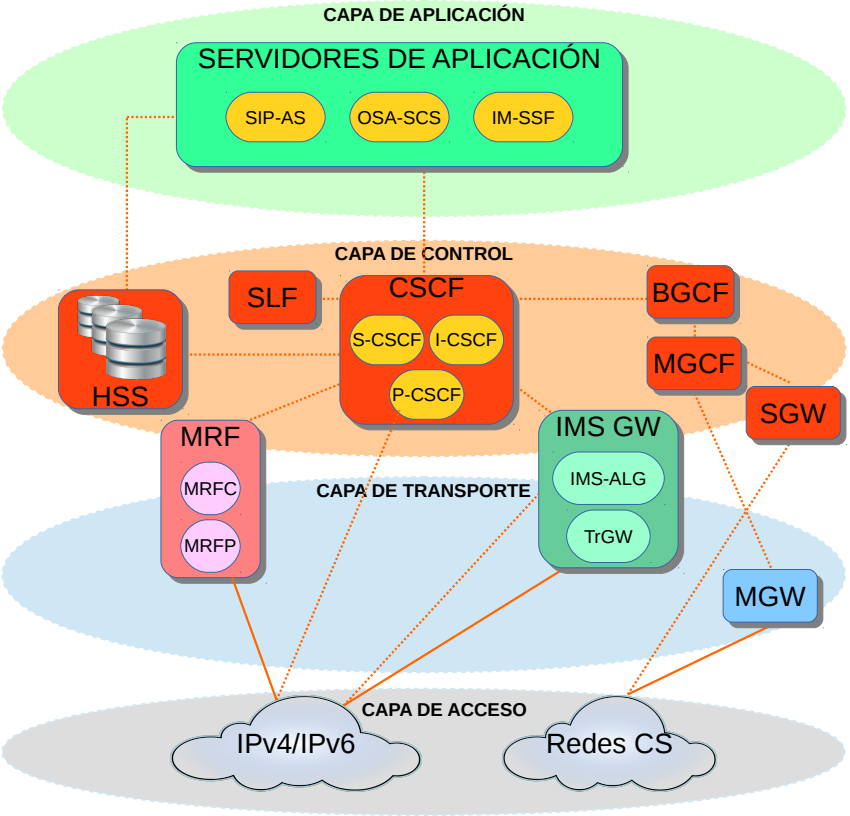


Figura 6.1: Arquitectura IMS

- **Capa de acceso**: Representa cualquier tipo de acceso como CDMA2000, redes de banda ancha, cable, WI-FI, etc. Deben disponer de capacidad para la conmutación de paquetes ya que la señalización de IMS se basa en este mecanismo. En el caso de tratarse de una red de conmutación de circuitos (PSTN por ejemplo) será necesaria la conversión de la señalización de circuitos a la señalización de conmutación de paquetes.
- **Capa de transporte**: Esta capa es donde residen los encaminadores y conmutadores junto con pasarelas que traducen protocolos entre redes conmutadas de circuitos y de paquetes.
- **Capa de control**: Es el núcleo del IMS y toma decisiones en base a políticas que se aplican en la capa de transporte. Esta capa proporciona el control y la gestión de sesiones y es responsable de su establecimiento y finlización. También contiene información acerca de la autenticación del abonado, la autorización de servicio y la ubicación.
- **Capa de servicio o aplicación**: Hospeda los servicios de aplicación y contenido, incluyendo los servidores de aplicaciones y servidores Web. También incluye habilitadores de servicios genéricos que gestionan elementos de servicio tales como grupos de usuarios y presencia. Estos elementos interconectan a los abonados a través del plano de control. La capa de aplicación es compatible con la mayoría de las aplicaciones multimedia o habilitadores de aplicación, como la presencia y la ubicación del abonado.

A continuación veremos las principales funciones que conforman la arquitectura IMS.

6.2.1. CSCF

El CSCF (Call/Session Control Function) es un conjunto de nodos esencial en la arquitectura IMS ya que procesan toda la señalización SIP. Hay tres tipos de CSCF dependiendo de la funcionalidad que proporcionan. Todos ellos se conocen colectivamente como CSCF pero cualquier CSCF pertenece a una de las siguientes tres categorías:

- P-CSCF (Proxy-CSCF)
- I-CSCF (Interrogating-CSCF)
- S-CSCF (Serving-CSCF)

P-CSCF (Proxy-CSCF)

Es el primer punto de contacto (en el plano de señalización) del terminal de los suscriptores con la red IMS. Desde la perspectiva de señalización SIP actúa como un Proxy Server, esto quiere decir que toda la señalización SIP generada por el terminal o destinada a él atraviesa el P-CSCF. Incluye diversas funciones:

- Valida la exactitud de los mensajes SIP y verifica que las peticiones son conforme a la norma.
- Garantiza la seguridad de los mensajes entre los UE y la red IMS utilizando asociaciones basadas en IPsec o TLS.
- Autentica la identidad del UE.
- Incluye un compresor/descompresor de mensajes SIP.

Puede incluir PDF (Policy Decision Function) que autorizará el acceso a los recursos del plano de aplicación y gestionará la calidad de servicio (QoS). También puede generar información de tarificación.

El número de nodos P-CSCF dependerá, por razones de escalabilidad y facilidad de integración, del número de usuarios a los que haya que atender.

I-CSCF (Interrogating-CSCF)

El I-CSCF es un proxy SIP situado en el borde de un dominio administrativo de IMS. Su dirección IP se publica en el Sistema de Nombres de Dominio (DNS) del dominio utilizando registros de tipo NAPTR y SRV para que los servidores remotos pueden localizarlo y utilizarlo como punto de entrada.

Implementa un interfaz de conexión con el HSS (Home Subscriber Server) mediante el uso del protoclo Diameter (RFC3588[1]) por la cual consulta al HSS la dirección del S-CSCF asignado a un UE para realizar el registro SIP o para el encaminamiento de peticiones hacia él.

Al ser un proxy SIP, el I-CSCF es el punto conocido por la red externa e indica el siguiente salto para llevar a cabo la señalización. Además, el I-CSCF puede cifrar partes de los mensajes SIP ocultando cualquier tipo de información sensible.

Opcionalmente, realiza también una función de ocultación hacia las redes externas, de manera que los elementos no IMS no pueden averiguar cómo se trata la señalización de forma interna (por ejemplo, el número, el nombre y la capacidad de la CSCF).

S-CSCF (Serving-CSCF)

El S-CSCF es el nodo central del plano de señalización en la red IMS. Además de la funcionalidad del servidor SIP también actúa como un SIP registrar. Esto es, mantiene una relación entre la localización del usuario (la IP del terminal desde el que accede) y la dirección SIP del usuario (conocida como Public User Identity). El S-CSCF también es responsable de procesar el registro de la ubicación de cada equipo de usuario, la autenticación y el encaminamiento de peticiones.

Toda la señalización de los terminales IMS atraviesa el S-CSCF asignado, el cual inspecciona y determina qué servidores de aplicación se deben visitar o si la señalización SIP debe visitar uno o más sistemas autónomos en su camino hacia el destino final.

Al igual que el I-CSCF, dispone de una interfaz gobernada por Diameter para la comunicación con el HSS desde la que:

- Accede a la información de autenticación del usuario que está intentando acceder a IMS.
- Accede al perfil de usuario. Este perfil contiene información útil para comprobar si el usuario está autorizado a un determinado servicio y en caso afirmativo saber, por cada servicio, dónde encaminar la petición SIP generada por el usuario.
- Informa al HSS que éste es el S-CSCF asignado al usuario mientras dure el registro.

El número de S-CSCF puede variar en cada red por razones de escalabilidad y redundancia. Cada S-CSCF atiende a un número de terminales IMS en función de la capacidad del nodo.

6.2.2. Bases de datos de usuario

El HSS (Home Subscriber Server) es un repositorio central para la información relacionada con los usuarios. Técnicamente se trata de una evolución del HLR (Home Location Register) de GSM.

Contiene toda la información de suscripción de los usuarios necesaria para gestionar sesiones multimedia. Esta información incluye, entre otras, información de localización, de seguridad (autenticación y autorización), perfil de usuario (indica los servicios a los que el usuario tiene acceso) y el S-CSCF donde se encuentra registrado.

Una red puede contener más de un HSS, sin embargo, todos los datos relacionados con un usuario en particular se almacenan en un único HSS. En caso que haya más de un HSS, se requiere de un SLF (Subscriber Location Function), que es una base de datos simple que almacena que HSS contiene la información de un usuario particular. Las interfaces de comunicación tanto del HSS como del SLF se implementan mediante el protocolo Diameter (RFC3588[**?**]).

6.2.3. Servidores de aplicación

Un Application Server (AS) es una entidad SIP que aloja y provee de servicios al IMS. Dependiendo del servicio en concreto puede actuar como un Proxy SIP, como SIP UA (User Agent) o SIP B2BUA (Back to Back User Agent). El AS puede estar situado en la "Home Network" o en una red de terceros, por ejemplo, en caso de un servicio que ofrezca un proveedor.

El AS se comunica con el S-CSCF y el I-CSCF mediante interfaces SIP y con el HSS mediante Diameter. Además, pueden implementar protocolos como HTTP con el fin de proporcionar una interfaz web de configuración a los terminales IMS. Hay tres tipos de AS:

- SIP AS (SIP Application Server): Hospeda y ejecuta Servicios Multimedia IP basados en SIP.
- OSA-SCS (Open Service Access - Service Capability Server): Este AS proporciona una interfaz para el marco OSA AS. Hereda todas las capacidades de OSA, especialmente la capacidad para acceder al IMS de forma segura desde redes externas. Este nodo actúa como un AS en un lado (interfaz SIP hacía el S-CSCF) y como una interfaz entre el OSA AS y la Interfaz de programación de aplicaciones OSA.
- IM-SSF (IP Multimedia Service Switching Function): Se trata de un servidor intermediario que puede actuar como un servidor de aplicaciones SIP y, al mismo tiempo, como un SSF (Service Switching Function) que interactúa con el gsmSCF mediante un protocolo basado en CAP (CAMEL Application Part) para conectar con los servicios CAMEL de redes GSM.

6.2.4. SBC

El SBC (Serial Border Controller), o también conocido como SBG (Serial Border Gateway), es el encargado de la correlación de toda la señalización y los flujos de media que pasa por los extremos de la red proporcionando un conjunto completo de funciones que son necesarias para acceder e interconectar el dominio IMS con otras redes IP multimedia.

Este nodo proporciona acceso con seguridad, protección del ancho de banda, calidad del servicio, nivel de servicios acordados y otras funciones críticas para las transmisiones en tiempo real de audio o vídeo. EL SBC se localiza en los extremos de la red, el punto de la infraestructura donde una sesión pasa de una red a otra. Podemos diferenciar dos partes que lo componen:

- SGC (Session Gateway Controller): Se encarga del plano de señalización.
- MG (Media Gateway): Soporta el tráfico de datos.

Existen tres grandes funciones para este nodo dentro de la red:

- A-SGC: cuando la funcionalidad SBG se implementa entre el core de IMS y la red de acceso. Sólo permite el tráfico de señalización hacia y desde los usuarios que están registrados en el HSS. La excepción se produciría con llamadas de emergencia de usuarios no registrados que pueden ser aceptadas si así se configura en el nodo.

- N-SGC: funcionalidad implementada entre el core de IMS y una red externa. Puede enrutar tráfico hacia cualquier red mediante descubrimiento DNS (DDDS) y los registros DNS NAPTR, SRV y A según indica el RFC3263[7] para redes SIP.
- MP (Media Proxy): Protege los nodos centrales de IMS de posibles ataques y bloquea el tráfico malicioso. Implementa también QoS para el control del ancho de banda.

6.2.5. MRF

EL MRF (Media Resource Function) implementa todas las funciones relacionadas con la media, tales como hacer sonar anuncios, mezclar flujos de medios, transcodificación, grabación, reconocimiento de voz, estadísticas, y en geneal, hacer cualquier tipo de análisis de medios.

Se divide en un nodo llamado MRFC (Media Resource Function Controller) dentro del plano de señalización y un nodo llamado MRFP (Media Resource Function Processor) dentro del plano de los medios de comunicación. El MRFC actúa como un Agente de Usuario SIP, contiene una interfaz SIP hacia el S-CSCF y controla los recursos del MRFP través de una interfaz H.248. El MRFP implementa las funciones relacionadas con los medios.

6.2.6. BGCF

El BGCF (Breakout Gateway Control Function) es un servidor SIP con capacidades de enrutamiento basado en números de teléfono. La función principal de este nodo es seleccionar la pasarela de salida de las solicitudes SIP dirigidas a un usuario en una red de conmutación de circuitos, tal como la PSTN. Por lo general, la BGCF utiliza una MGCF (Media Gateway Control Function) en la propia red de IMS para interactuar con la PSTN. Si el acceso a la PSTN debe ser enviado a otra red IMS, el BGCF envía la petición SIP al BGCF de la otra red.

6.2.7. Pasarelas IP

IMS es compatible con IP versión 4 (IPv4[5]) e IP versión 6 (IPv6[2]) por lo que en algún momento de una sesión o de la comunicación multimedia IP, se puede producir el interfuncionamiento entre las dos versiones. Con el fin de facilitarlo sin necesidad de soporte por parte de los terminales, IMS añade dos nuevas entidades funcionales que proporcionan la traducción entre ambos protocolos. Estas nuevas entidades son el IMS-ALG (IMS Application Layer Gateway) y el TrGW (Transition Gateway). El primero controla la señalización del plano de control (por ejemplo, SIP y mensajes SDP) mientras que el segundo se encarga del flujo media (por ejemplo, RTP, RTCP).

El IMS-ALG actúa como un B2BUA mediante el mantenimiento de dos patas de señalización independientes: una hacia la red IMS interna y la otra hacia la otra red. Cada una de estas patas se están ejecutando sobre una versión IP diferente. Además, el IMS-ALG reescribe el SDP cambiando las direcciones IP y números de puerto creados por el terminal con una o más direcciones IP y números de puerto asignados al TrGW forzando que el flujo media se encamine a través del éste.

El TrGW es un NAT-PT/NAPT-PT (Network Address Port Translator–Protocol Translator) que está configurado con un conjunto de direcciones IPv4 e IPv6 que se asignan dinámicamente a cada sesión.

6.2.8. Pasarelas PSTN/CS

Una pasarela hacia PSTN/CS proporciona una interfaz de comunicación con la red de conmutación de circuitos, permitiendo a los terminales IMS recibir y realizar llamadas a la red PSTN. Se compone de las siguientes funciones:

- SGW (Signaling Gateway): Conecta el plano de señalización de la red CS. Lleva a cabo la conversión de protocolo de capa inferior entre ISUP/MTP o BICC/MTP a ISUP o BICC sobre SCTP/IP.
- MGCF (Media Gateway Control Function): Es un sistema flexible que puede integrarse en diferentes tipos de soluciones y adaptarse a las necesidades de los operadores o usuarios finales. Mediante el uso de estándares abiertos como SIP, ISUP, H.248 y protocolos RTSP se pueden crear diferentes soluciones adaptadas a las diferentes necesidades.

- MGW (Media Gateway): Interconecta el plano de medios de la red PSTN o CS. Permite enviar y recibir datos a través del protocolo RTP en un extremo y uno o más PCM (Pulse Code Modulation) para conectarse a la red CS en el otro. También realiza la transcodificación de datos cuando el terminal IMS no soporta el códec utilizado en el dominio de la red de conmutación.

6.3. Protocolos en IMS

Existen diversos protocolos que se utilizan en IMS. Entre ellos destacan el Protocolo de Iniciación de Sesión (SIP), Diameter, SigComp, el protocolo de transporte en tiempo real (RTP), el protocolo de control RTP (RTCP) y el protocolo de seguridad IP (IPsec).

Dado que el protocolo SIP ya se ha estudiado en detalle, a continuación sólo se expone una breve descripción del protocolo Diameter que se utiliza en nuestro escenario.

6.3.1. DIAMETER

El protocolo Diameter es una evolución de protocolos de autenticación, autorización y contabilidad (AAA) como RADIUS, TACACS, Kerberos y COPS. RADIUS fue diseñado originalmente para redes punto a punto (PPP). Carece de escalabilidad, y por lo tanto no se puede implementar en grandes redes. Además, no es compatible con itinerancia que es el principal requisito de las redes móviles. RADIUS y TACACS tienen sus propias ventajas, pero no son adecuados para las redes de hoy en día.

Diameter fue desarrollado para cumplir estos requisitos. Además, admite el protocolo de autenticación extensible (EAP) que se usa comúnmente como un mecanismo de seguridad para evitar la autenticación fraudulenta. También ofrece control de crédito, validación de la identidad, políticas de acceso y funciones de tarificación para los clientes de la red.

Capítulo 7

Implementación de un sistema IMS

En este capítulo se expone el proceso de implementación del sistema IMS basado en Kamailio, FHoSS y Asterisk utilizado en este proyecto. La figura 7.1 muestra los elementos que componen dicho escenario.

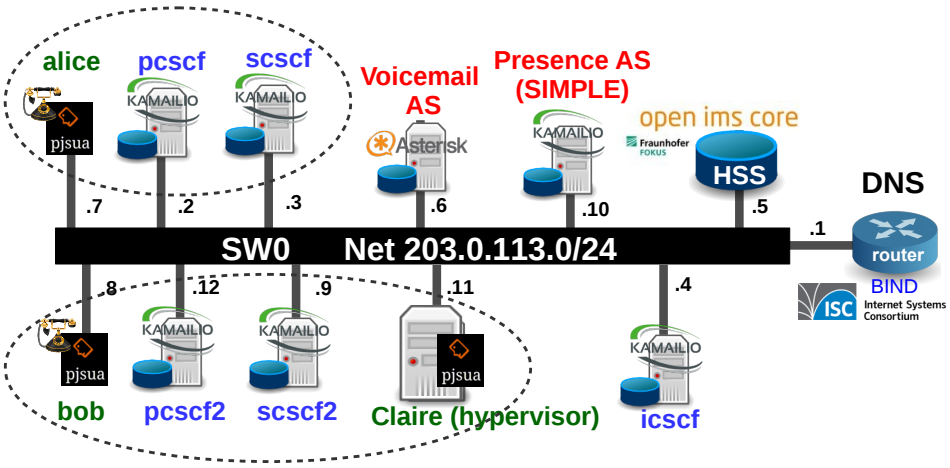


Figura 7.1: Escenario IMS.

- Un servidor DNS.
- Dos servidores Kamailio que ejecutan la función P-CSCF.
- Un servidor Kamailio que ejecuta la función I-CSCF.
- Dos servidores Kamailio que ejecutan la función S-CSCF.
- Un servidor que ejectua la función del HSS.
- Un servidor Asterisk que ejecuta la función de un AS para el Voicemail.
- Un servidor Kamailio que ejecuta la función de un AS para Presencia.
- Dos hosts en los que se configuran los clientes SIP PJSUA.
- Un PC anfitrión en el que se ejecuta el escenario, un cliente SIP PJSUA y se realizan las capturas de tráfico.

El fichero de definición del escenario (Anexo C.1.1) contiene toda la información relativa a los elementos anteriores además de órdenes de ejecución específicas que iremos analizando en cada sección de este capítulo.

7.1. CSCF

En este proyecto se han implementado las diferentes funciones CSCF mediante servidores Kamailio. Tomando como base la configuración específica que Kamailio ha desarrollado para cada uno de los elementos que lo componen, se han hecho algunas modificaciones que veremos más adelante para adaptarla a las particularidades del escenario.

7.1.1. Instalación y configuración del I-CSCF

Instalamos Kamailio con los módulos específicos de ims, mysql y los módulos de conexión de Kamailio con mysql:

```
# apt-get install kamailio kamailio-ims-modules kamailio-presence-modules kamailio-tls-modules \
kamailio-xml-modules mysql-server kamailio-mysql-modules kamailio-utils-modules
```

Kamailio distribuye un fichero con las instrucciones para crear la estructura de la base de datos del I-CSCF. La información más relevante que contiene esta base de datos es el dato actualizado del último S-CSCF en el que se ha registrado cada usuario. Esto permite al I-CSCF informar al HSS de los S-CSCF activos en el sistema.

Así mismo, la base de datos local debe alimentarse con la información de los S-CSCF disponibles (dirección IP o nombre DNS y puerto o puertos de servicio). Esta información permitirá al I-CSCF asignar un S-CSCF en caso que el HSS no provea esta información.

Con objeto de alimentar esta base de datos con los datos propios de los S-CSCFs existentes, se ha modificado el fichero de creación de la base de datos con dicha información como se muestra en el anexo C.2.1. Además, se han creado dos scripts que automatizan la generación de la base de datos y que se ejecutan al iniciar el escenario (Anexos C.1.3 y C.2.2).

Kamailio distribuye un fichero de configuración específico para este nodo que en nuestro escenario es válido y no ha sido necesario modificar. La configuración específica del nodo (dirección IP, dominio...) se encuentra en un fichero a parte, “icscf.cfg”, cuyo contenido puede consultarse en el anexo C.2.3. Así mismo, el anexo C.2.4 contiene la configuración del dominio SIP de servicio y la configuración del motor para la base de datos, en este caso MYSQL.

En cuanto a la comunicación con el HSS, interfaces Cx de Diameter, se han configurado los parámetros en el fichero “icscf.xml” cuyo contenido podemos ver en el anexo C.2.5.

Por último, es necesario configurar la variable “RUN_KAMAILIO” con el valor “yes” en el fichero /etc/default/kamailio, anexo C.2.6, para permitir el inicio del proceso. En este mismo fichero de configuración podemos modificar el usuario y grupo de ejecución del proceso, la cantidad de memoria reservada de los procesos o incluso modificar el path y nombre del fichero de configuración. En este escenario se han mantenido los valores por defecto.

7.1.2. Instalación y configuración de P-CSCF

Instalamos Kamailio con los módulos específicos de ims, mysql y los módulos de conexión de Kamailio con mysql:

```
# apt-get install kamailio kamailio-ims-modules kamailio-presence-modules kamailio-tls-modules \
kamailio-xml-modules mysql-server kamailio-mysql-modules kamailio-utils-modules
```

Kamailio distribuye un fichero con las instrucciones para crear la estructura de la base de datos del P-CSCF (C.3.1). A diferencia del I-CSCF, esta base de datos sólo se utiliza con datos en tiempo real relativa a la localización de los usuarios por lo que no es necesario añadir ninguna información adicional al fichero original.

Si se han creado dos scripts que automatizan la generación de la base de datos y se ejecutan al iniciar el escenario cuyo contenido podemos ver en los anexos C.1.3 y C.3.2. También se ha creado un script, anexo C.3.3, junto con un fichero que contiene un volcado de la base de datos recién creada con objeto de permitir la restauración de la base de datos a un punto inicial mediante el uso de etiquetas vnuml.

En cuanto a la configuración, Kamailio distribuye un fichero de configuración específico que en nuestro escenario ha sido necesario modificar (Anexo C.3.4):

```
#       LAB - Modificamos el routing para que los usuarios de la plataforma puedan recibir llamadas...
#       if (!ds_is_from_list()) { [1]
#               # Originating from Subscriber:
#               route(Orig_Initial);
#       } else {
#               # Terminating to Subscriber:
#               route(Term_Initial);
#       }

        if ($route_uri =~ "sip:term@"+".*") { [2]
                # Terminating to Subscriber:
                route(Term_Initial);
        } else {
                # Originating from Subscriber:
                route(Orig_Initial);
        }
# Fi LAB
```

- **[1]** La función "ds_is_from_list()" devuelve positivo si el mensaje SIP proviene de un elemento del listado de "dispatchers". En este escenario esta lista está vacía dado que se ha intentado que el sistema no contenga, en la medida de lo posible, datos locales referentes a la arquitectura del IMS.
- **[2]** Durante el proceso de registro, el P-CSCF añade la cabecera **`Path: term@<alias_p-cscf>;lr`** para indicar que debe estar en el camino de sucesivas peticiones. Esta nomenclatura además permite identificar el último salto SIP antes de llegar al UA destino. Así, en el código modificado, lo que indicamos es que si la primera cabecera **`Route`** de la petición sigue dicha nomenclatura, estamos en el último salto SIP y el mensaje SIP tiene como destino un usuario del propio P-CSCF. Por tanto, se debe invocar a la rutina específica correspondiente. En cualquier otro caso, se una petición inicial, es decir, no procedente de un elemento de la plataforma IMS si no de un suscriptor de ese proxy.

Es decir, la configuración inicial generaba que las peticiones con destino un usuario de la plataforma, una vez ya habían sido encaminadas por el S-CSCF correspondiente, fueran rechazadas dado que se volvían a evaluar como peticiones procedentes de un usuario que, generalmente, ni pertenece a ese proxy por lo que la petición se rechaza.

La configuración específica del nodo (dirección IP, dominio...) se encuentra en un fichero separado: pcscf.cfg. El contenido de ambos ficheros se puede consultar en el anexo C.3.5 para el servidor pcscf.example.org y C.3.6 para el servidor pcscf2.example.org.

No ha sido necesario configurar las interfícies Diameter puesto que el P-CSCF se comunica a través de Diameter con el PCRF en caso de aplicar políticas. En este escenario no se ha configurado ningún PCRF por lo que no se ha activado el módulo de Diameter en el Proxy (opción WITH_RX).

Por último, es necesario configurar la variable "RUN_KAMAILIO" con el valor 'yes" en el fichero /etc/default/kamailio, anexo C.3.7, para permitir el inicio del proceso. En este mismo fichero de configuración podemos modificar el usuario y grupo de ejecución del proceso, la cantidad de memoria reservada para los proceso de Kamailio o incluso modificar el path y nombre del fichero de configuración. En este escenario se han mantenido los valores por defecto.

7.1.3. Instalación y configuración de S-CSCF

Instalamos Kamailio con los módulos específicos de ims, mysql y los módulos de conexión de Kamailio con mysql:

```
apt-get install kamailio kamailio-ims-modules kamailio-presence-modules kamailio-tls-modules \
kamailio-xml-modules mysql-server kamailio-mysql-modules kamailio-utils-modules
```

Kamailio distribuye un fichero con las instrucciones para crear la estructura de la base de datos del S-CSCF (Anexo C.4.1). Al igual que en el caso del P-CSCF, esta base de datos sólo se alimenta con datos en tiempo real relativos a la localización y registro de los usuarios por lo que no es necesario añadir ninguna información adicional al fichero original.

Si se han creado dos scripts que automatizan la generación de la base de datos y se ejecutan al iniciar el escenario cuyo contenido podemos ver en los anexos C.1.3 y C.4.2. También se ha creado un script, anexo C.4.3, junto con un fichero que contiene un volcado de la base de datos recién creada con objeto de permitir la restauración de la base de datos a un punto inicial mediante el uso de etiquetas vnuml.

La configuración del S-CSCF difiere de la general. Kamailio distribuye un fichero de configuración específico que en nuestro escenario ha sido necesario modificar para incorporar la funcionalidad del voicemail en caso de que el usuario final no conteste, cuelgue o se produzca un error al procesar la llamada (Anexo C.4.4).

La primera modificación se ha hecho en la rutina de Kamailio que procesa las llamadas hacia el usuario terminal (rutina "term") en la que hemos introducido una nueva rutina a ejecutar en caso que la petición sea de tipo INVITE y se reciba o se genere un código de error como respuesta:

```
        if (is_method("INVITE")) {
                #Introducimos la ruta al Voicemail
                # en caso de error
                if(!t_is_set("failure_route")) t_on_failure("VOICEMAIL");
                                ...
```

También se ha configurado un timeout de 8 segundos para usuarios finales registrados mientras que el timeout general es de 10 segundos.

```
                                ...
                #Modificamos el timeout de llamada a 8 segundos
                #para usuarios registrados en la plataforma
                if (impu_registered("location")) {
                        t_set_fr(8000);
                }
        }
```

Por último, se ha configurado la rutina "VOICEMAIL"que reencamina estas llamadas al servicio de buzón de voz siempre y cuando se trate de llamadas hacia un usuario registrado:

```
#LAB - Ruta al Voicemail
failure_route[VOICEMAIL] {
        xlog("L_ERR","User unavailable, voicemail route forced\n\n");
        if (impu_registered("location")) {
                rewritehostport("voicemail.example.org:5060");
                t_relay();
        }
}
#Fi LAB
```

En el anexo A se puede encontrar una explicación más detallada de las funciones involucradas en el código.

La configuración específica del nodo (dirección IP, dominio, módulos activos...) se encuentra en un fichero separado: scscf.cfg. Como hemos mencionado al inicio del capítulo, el escenario dispone de dos servidores S-CSCF. Este fichero de configuración difiere entre ambos servidores. El contenido de ambos ficheros se puede consultar en el anexo C.4.5 para el servidor scscf.example.org y el anexo C.4.6 correspondiente a scscf2.example.org.

Los S-CSCF interaccionan con el HSS mediante el protocolo DIAMETER. Los anexos C.4.7 y C.4.8 contienen respectivamente la configuración de las interfícies Cx de comunicación con el HSS del S-CSCF y S-CSCF2.

Por último, al igual que en los apartados anteriores, es necesario configurar la variable “RUN_KAMAILIO” con el valor ‘yes” en el fichero /etc/default/kamailio, anexo C.4.9, para permitir el inicio del proceso. Se han mantenido el resto de valores por defecto.

7.2. HSS

El núcleo de nuestro IMS quedaría incompleto sin un Home Subscriber Server (HSS) cuyo propósito es el de proporcionar una base de datos centralizada de usuarios para gestión de autorización y capacidades. Es ante todo el nexo de unión entre un sistema de gestión de base de datos y las interfaces Diameter del CSCF y las aplicaciones.

FOKUS ha desarrollado su propio prototipo de HSS implementado íntegramente en Java y basado en open source. FHoSS almacena los perfiles de usuario y proporciona información de ubicación. Además, también dispone de una consola de administración basada en acceso web. La figura 7.2 ilustra los principales componentes y las interfaces que se utilizan en él.

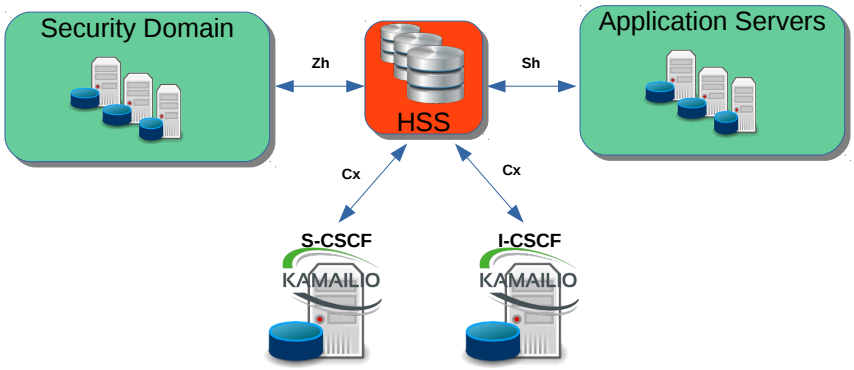


Figura 7.2: Esquema FHoSS.

De la figura anterior, las entidades que se comunican con FHoSS son el servidor de aplicaciones (AS) que aloja servicios en el entorno IMS y los Call State Control Function servers (CSCF). FHoSS es capaz de almacenar ficheros de usuario en servidores de aplicaciones a través de la interfaz Sh, y se comunica con los elementos del CSCF a través de interfaces Cx.

7.2.1. Interface layer

El núcleo de FHoSS está implementado en la clase Java HssDiameterStack. Además, utiliza la clase DiameterPeers para enviar peticiones a otras entidades y recibe peticiones a través de la clase CommandListener. Hay tres interfaces utilizadas en FHoSS, que son Sh, Cx, y Zh cada una de ellas con una implementación específica.

En el escenario del estudio, los servidores de aplicaciones están basados en Asterisk para el AS de voicemail y Kamailio para el de presencia. Asterisk no dispone de capacidades para la implementación de la interfaz Sh por lo que el HSS sólo utiliza las interfaces Cx.

7.2.2. Capa de acceso a los datos

Los datos operativos del FHoSS se almacenan en una base de datos MySQL. Sin embargo, implementa una capa de acceso a los datos (DAL) basada en JDBC lo que permite la comunicacón con cualquier servidor de base de datos siempre y cuando disponga de un conector JDBC. En este proyecto, MySQL es el motor de base de datos escogido.

7.2.3. GUI

Para gestionar y mantener la base de datos, se proporciona una interfaz de gestión basada en web basado en tomcat.

7.2.4. Instalación

Obtener el código fuente

Como punto de partida, el código fuente de FHoSS se encuentra disponible en el enlace:

```
https://svn.code.sf.net/p/openimscore
```

El código fuente está pre-configurado para trabajar desde una ruta específica. Por ello, en primer lugar, debemos crear el directorio "/opt/OpenIMSCore" y descargar el código fuente en éste:

```
# mkdir /opt/OpenIMSCore
# cd /opt/OpenIMSCore
# svn checkout https://svn.code.sf.net/p/openimscore/code/FHoSS/trunk FHoSS
```

Compilar e instalar

Antes de la compilación, se debe disponer de JDK >= 1.5 además de otras dependencias que se muestran a continuación:

```
# apt-get install subversion bison libcurl4-dev debhelper cdbs lintian build-essential fakeroot devscripts \
pbuilder dh-make ant openjdk-6-jdk libxml2 libxml2-dev*
```

Compilamos el código fuente:

```
# cd /opt/OpenIMSCore/FHoSS
# ant compile
```

Y por último, instalamos:

```
# ant deploy
```

Configuración del entorno

Con el fin de preparar la estructura de la base de datos que FHoSS utilizará, la propia distribución incorpora dos ficheros con las secuencias SQL necesarias para su creación:

- hssdb.sql: Contiene las secuencias relativas a la creación de la base de datos y las tablas necesarias.
- userdata.sql: Contiene información de usuarios de ejemplo.

Con el fin de disponer de un punto de partida, durante la primera ejecución del escenario se creó un script que automatiza la generación de la base de datos del HSS y que se puede consultar en el anexo C.5.1.

Este script crea la base de datos del HSS a partir del fichero de definición hss_db.sql y genera la vista que consultará el AS de Voicemail y que veremos en detalle en el apartado 7.4. El contenido del fichero hss_db.sql no se incluye en los anexos dada su extensión y que no se han realizado cambios sobre el mismo.

Se ha prescindido del fichero userdata.sql ya que la configuración de usuarios y de servicios se realiza a través del entorno web durante la ejecución de las prácticas.

Sin embargo, se han realizado varios volcados de la base de datos con el fin de disponer de diferente puntos de restauración. En todos ellos, se ha creado un script que regenera el estado inicial de la base de datos y, a continuación, importa un fichero de datos indicando en el argumento. El script de regeneración puede consultarse en el anexo C.5.2.

Los puntos de restauración generados son:

- presence_ini.sql (Anexo C.5.3): Inicio de la práctica de Presencia. Se crean de forma automática los usuarios Alice, Bob y Claire y los S-CSCF.
- voicemail_ini.sql (Anexo C.5.4): Inicio de la práctica de Voicemail. Se crean de forma automática los usuarios Alice, Bob y Claire, los S-CSCF y el servicio de Presencia (AS, iFC, y TP) dentro del perfil de servicio ims_services.
- voicemail_route.sql (Anexo C.5.5): Inicio de la segunda parte de la práctica de Voicemail. Se crean de forma automática los usuarios Alice, Bob y Claire, los S-CSCF y los servicios de Presencia y Voicemail (AS, iFC, y TP) dentro del perfil de servicio ims_services. En este punto, el servicio de Voicemail todavía no incluye el TP de consulta del buzón de voz de los usuarios.
- complete.sql (Anexo C.5.6): Escenario final. Crea de forma automática los usuarios Alice, Bob y Claire, los S-CSCF y los servicios de Presencia y Voicemail (AS, iFC, y TP) dentro del perfil de servicio ims_services. En este punto, el servicio de Voicemail ya incluye el TP de consulta del buzón de voz de los usuarios.

Configuración del core

Ya tenemos creada la base de datos por lo que lo que nos queda es echar un vistazo a los archivos de configuración del FHoSS y modificar algunos parámetros según sea apropiado para nuestra instalación.

- DiameterPeerHSS.xml (Anexo C.5.7): Contiene la configuración de las interfaces de Diameter. En este fichero hemos configurado el FQDN de nuestro HSS, el dominio y el puerto por el que acepta conexiones. También hemos configurado los peers de Diameter, es decir, FQDN, dominio y puerto para la comunicación con el i-cscf y los s-cscf.
- hibernate.properties (Anexo C.5.8): Contiene la configuración de la conexión a la base de datos: IP, puerto, usuario, password y tipo de conector, por defecto mysql. Se han mantenido los valores por defecto.
- hss.properties (Anexo C.5.9): Especificamos la dirección IP y puerto por la que escucha el servicio del tomcat y el intervalo de tiempo en el que el HSS revisará si hay cambios en la BBDD que deba informar a los s-cscf. Se han modificado los siguientes parámetros:

```
#IP de servicio del Tomcat
host=203.0.113.5
#Puerto de servicio del Tomcat
port=8080
#Intervalo de revisión de cambios en BBDD y notificación a través de las interfaces Cx (en segundos):
CX_EVENT_CHECK_INTERVAL=1
```

- Log4j.properties (Anexo C.5.10): Contiene configuración para el proceso de log: donde se guardan y el nivel de debug. Se han mentenido los valores por defecto.

Inicio automático

Durante la realización del proyecto se observó que el servidor FHoSS debía ser iniciado manualmente a través del script /opt/OpenIMSCore/FHoSS/deploy/startup.sh.

Además, este script no dispone de las rutas correctas a las librerías de java ni al propio binario.

Con objeto de iniciar de forma automática el servicio al inicio del sistema operativo, se ha adaptado el script de arranque publicado por Gaoithe en Github (anexo C.5.11), se ha configurado un script intermedio también publicado por Gaoithe (anexo C.5.12), y se ha adaptado al escenario el script original de inicio (anexo C.5.13) con el camino correcto al binario de java.

Configuración general de servicio

Accedemos a la consola de administración mediante un navegador web en http://203.0.113.5:8080 con el usuario administrador que hemos definido en el fichero de configuración tomcat-users.xml (anexo C.5.14).

En el apartado "Network Configuration"debemos especificar los datos generales de nuestra plataforma. En contreto:

```
Visited Networks: example.org
Preferred S-CSCF Set: Es un listado de los S-CSCF de la plataforma

Preferred S-CSCF Set 1: scscf.example.org
        Priority 0: scscf.example.org:5060
        Priority 1: scscf2.example.org:5060

Preferred S-CSCF Set 2: scscf2.example.org
        Priority 0: scscf2.example.org:5060
        Priority 1: scscf.example.org:5060
```

Creación de los abonados

Cada usuario de IMS (IMSU) dispone de al menos una identidad privada y una o más identidades públicas.

La identidad privada en IMS (IMPI) sigue la estructura NAI:

```
alice@example.org
```

La identidad de usuario pública (IMPU) utiliza el formato de SIP URI o TEL URI:

```
sip:alice@example.org
tel:*86
```

Las figuras 7.3, 7.5 y 7.4 muestran los detalles de la configuración del abonado Alice.

```
Usuario1:
IMSU: alice
IMPI: alice@example.org
Contraseña: alice
Preferred S-CSCF: scscf.example.org
Esquema de autenticación por defecto: Digest-MD5
AMF:0x00..0
OP:0x00..0
SQN:0x00..0
IMPU: sip:alice@example.org
Service Profile: ims_services
IMPU Type: Public_User_Identity
List of Visited Networks: example.org
```

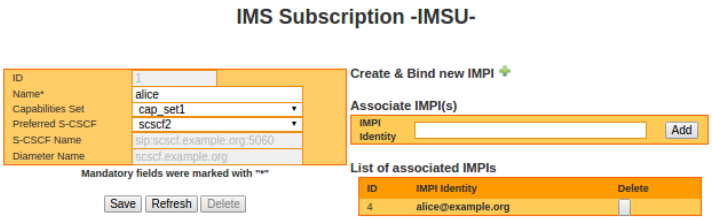


Figura 7.3: Identidad IMS (IMSU) de Alice.

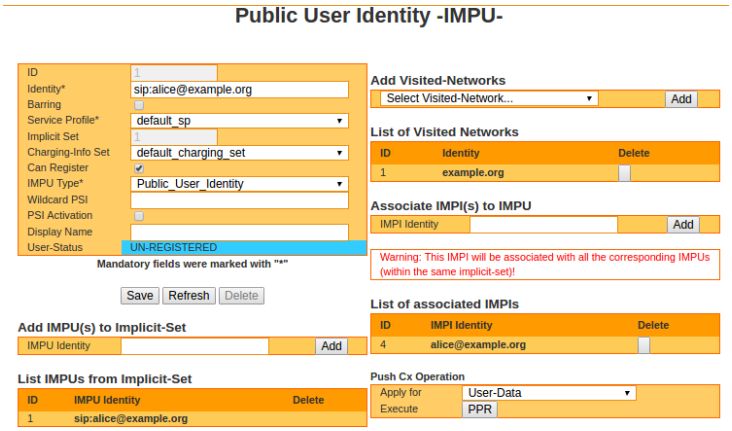


Figura 7.4: Identidad pública (IMPU) de Alice.

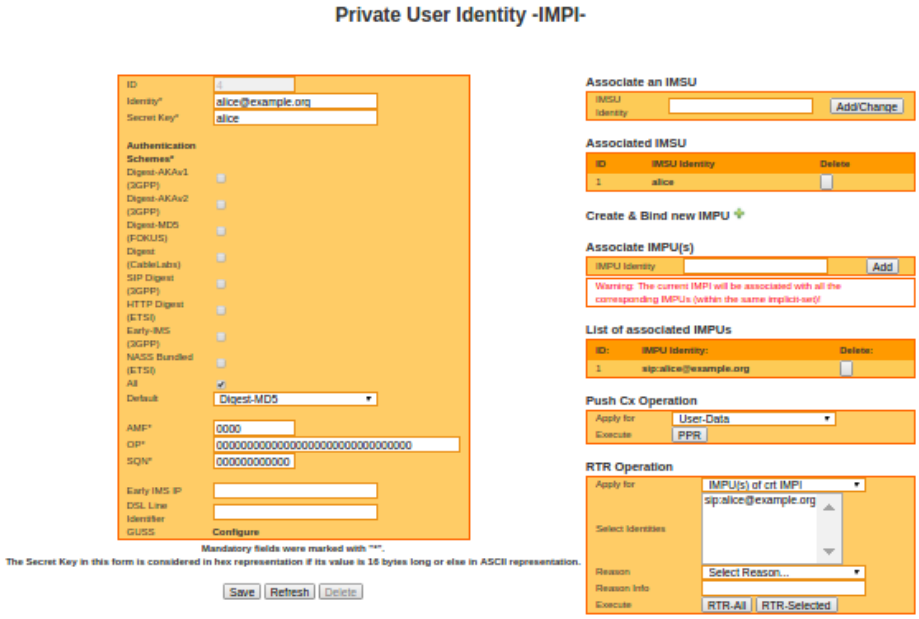


Figura 7.5: Identidad privada (IMPI) de Alice.

A continuación se muestran los detalles del resto de abonados:

```
Usuario2:
IMSU: bob
IMPI: bob@example.org
Contraseña: bob
Preferred S-CSCF: scscf2.example.org
Esquema de autenticación por defecto: Digest-MD5
AMF:0x00..0
OP:0x00..0
SQN:0x00..0
IMPU: sip:bob@example.org
Service Profile: ims_services
IMPU Type: Public_User_Identity
List of Visited Networks: example.org
```

```
Usuario3:
IMSU: Claire
IMPI: claire@example.org
Contraseña: claire
Preferred S-CSCF: scscf2.example.org
Esquema de autenticación por defecto: Digest-MD5
AMF:0x00..0
OP:0x00..0
SQN:0x00..0
IMPU: sip:claire@example.org
Service Profile: ims_services
IMPU Type: Public_User_Identity
List of Visited Networks: example.org
```

Durante la realización del proyecto se observaron problemas de negociación de los retos de seguridad en diferentes esquemas de validación. Es por ello que todos los usuarios de la plataforma se han configurado con el esquema por defecto “Digest-MD5”, esquema con el que el cliente pjsua completa el reto correctamente. Se han habilitado el resto por compatibilidad con otros posibles clientes SIP en un futuro.

Se han realizado varias exportaciones de la base de datos a diversos ficheros de texto que permitirán crear los servicios y usuarios de forma automatizada mediante el uso de etiquetas vnuml.

Configuración de los servicios

En el apartado anterior hemos visto que los abonados de la plataforma tienen asociado un "Service Profile".

Este perfil permite configurar como debe responder la plataforma antes eventos específicos como peticiones de presencia o desvío de llamadas al buzón de voz.

Un Service profile no es más que un listado de IFCs (Initial Filter Criteria) que, a su vez, es la asociación de un TP (Trigger Point) con un AS (Application Server).

Es decir, define que, bajo el cumplimiento de unos eventos específicos, la petición SIP debe encaminarse a un AS específico.

En este proyecto se ha configurado un único perfil de servicio como muestra la figura 7.6 llamado “ims_services” asociado a todos los suscriptores y que contiene dos IFCs:

- Presence: Determina a qué AS deben encaminarse los eventos de presencia.
- Voicemail: Determina bajo que condiciones debe encaminarse una llamada (INVITE) al AS del buzón de voz y define la uri sip:*86@example.org como uri destino con la que los usuarios de la plataforma podrán consultar su propio buzón de voz.

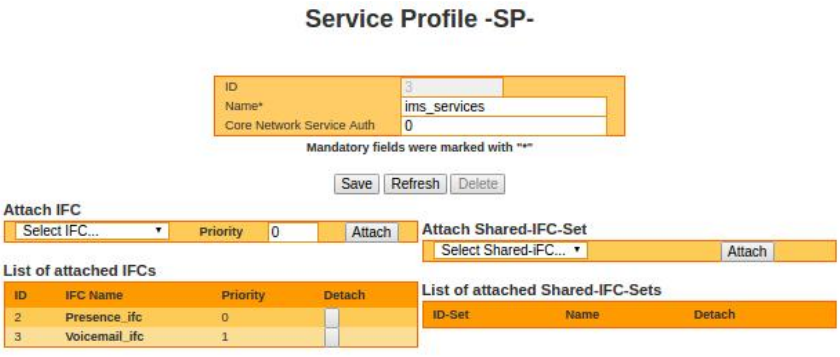


Figura 7.6: Service Profile: ims_services.

El AS de presencia tiene las siguientes características:

```
Name: Presence
Server Name: sip:presence.example.org:5060
Diameter FQDN: presence.example.org
Default Handling: Session - Continued
```

El TP de presencia por su parte es una OR de las siguientes condiciones:

```
Condición 1:
[Sip Method: PUBLISH] and [[Sip Header: Event] and [Sip Header Content: .*presence.*]]
and [Session Case: Origin - Session]

Condición 2:
[Sip Method: PUBLISH] and [[Sip Header: Event] and [Sip Header Content: .*presence.*]]
and [Session Case: Origin - UnReg]

Condición 3:
[Sip Method: SUBSCRIBE] and [[Sip Header: Event] and [Sip Header Content: .*presence.*]]
and [Session Case: Term - Reg]

Condición 4:
[Sip Method: SUBSCRIBE] and [[Sip Header: Event] and [Sip Header Content: .*presence.*]]
and [Session Case: Term - UnReg]
```

Las características del servicio de buzón de voz se muestran en las figuras 7.7, 7.8 y 7.9.

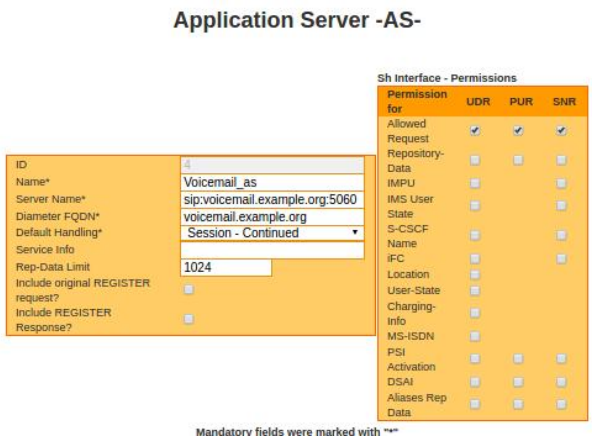


Figura 7.7: Definición del AS de Voicemail.

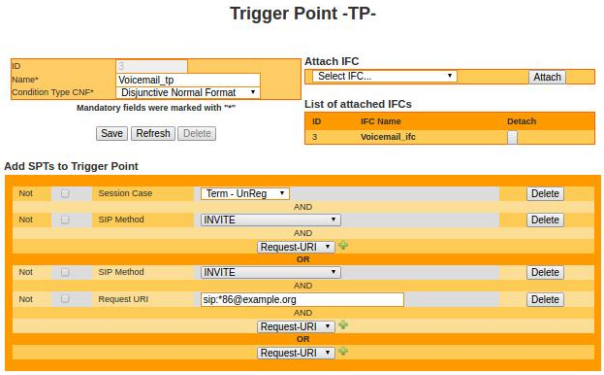


Figura 7.8: Trigger points del AS de Voicemail.

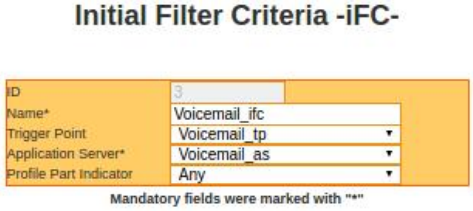


Figura 7.9: Initial Filter Criteria para el AS de Voicemail.

Dado que los S-CSCF procesan los iFCs antes de ejectuar cualquier otra rutina, hemos definido un trigger point que indica que si el método recibido es un INVITE y el usuario destino no se encuentra registrado (y por tanto no se encuentra disponible) o el usuario destino es la uri sip:*86@example.org, la llamada debe encaminarse al buzón de voz.

Existes tres posibles estados de los usuarios en cuanto a su registro:

- Not-Registered: El usuario no se encuentra registrado ni tiene un S-CSCF asignado. Esta condición se da cuando un usuario no ha realizado ningún registro en la plataforma. Cuando el I-CSCF obtiene este estado del HSS para un usuario, envía la petición SIP a uno de los S-CSCF que tiene es su base de datos y será éste el que procese el iFC desviando la llamada al buzón de voz.
- Unregistered: El usuario no se encuentra registrado pero tiene un S-CSCF asignado. Esta condición aparece cuando un usuario ha hecho un registro y un desregistro previo. Cuando el I-CSCF obtiene este estado del HSS para un usuario, envía la petición SIP al S-CSCF informado por el HSS y será éste el que procese el iFC y desvíe la llamada al buzón de voz.
- Registered: El usuario se encuentra registrado en la plataforma y tiene un S-CSCF asignado. Bajo esta circunstancia no aplica el iFC del voicemail.

Ante la segunda condición de este trigger point, el servidor Asterisk se ha configurado, como veremos en el apartado 7.4, para que inicie el buzón de voz del usuario origen en modo consulta. Es decir, es la condición que permite a los usuarios de la plataforma consultar su propio buzón de voz.

En caso que el usuario destino no responda, cuelgue, o se produzca algún error, son las propias rutinas que hemos desarrolado en los S-CSCF las que se encargarán de desviar el INVITE al buzón de voz.

7.3. AS: Presencia

Se instala un servidor Kamailio con los módulos de presencia y una base de datos local mysql que hará las funciones de servidor de aplicaciones para el servicio de presencia:

```
# apt-get install kamailio kamailio-presence-modules kamailio-xml-modules \
kamailio-mysql-modules kamailio-utils-modules mysql-server
```

El fichero kamailio-local.cfg, que podemos consultar en el anexo C.6.1, contiene la configuración específica del nodo como la IP, puerto y protocolo de servicio, los módulos de presencia y mysql y las opciones de resolución DNS. Remarcar que este Kamailio se ha configurado sin el módulo de autenticación activado ya que éste recibirá peticiones SIP de presencia de usuarios que previamente han conseguido registrase (y por tanto ya han pasado la fase de autenticación).

En el fichero kamctlrc, anexo C.6.2, hemos configurado el dominio SIP y el tipo de base de datos. Ello nos permite automatizar la creación de la base de datos local mediante el script general de creación de la base de datos de Kamailio, anexo C.1.3.

También se ha creado un script y un punto de restauración de la base de datos que puede ejecutarse mediante cualquiera de las etiquetas vnuml definidas. El contenido de dicho script podemos consultarlo en el anexo C.6.3. No se ha incluído el fichero sql con el punto de restauración (/tmp/kam_pras_restore.sql) debido a que no se han realizado modificaciones respesto a la estructura original de la base de datos de Kamailio.

7.4. AS: Voicemail

Se instala un servidor asterisk que hará las funciones de buzón de voz.

Proceso de instalación:

```
# apt-get install libmysqlclient-dev libmysqlclient15-dev autotools-dev libbluetooth-dev
# apt-get install asterisk debhelper asterisk-dev
# wget https://launchpad.net/ubuntu/+archive/primary/+files/asterisk-addons_1.6.2.1.orig.tar.gz
# tar xvjf asterisk-addons_1.6.2.1.orig.tar.gz ; cd asterisk-addons_1.6.2.1.orig
# make clean;  ./configure ; make ; make install; make samples
```

Para conseguir la integración con IMS, se configura el servidor Asterisk en Realtime de forma que se alimente de la información almacenada en el HSS.

Al tratarse de una instalación en Realtime, necesitaremos una base de datos de la que recoger la configuración de nuestra PBX y nuestros voicemails.

En este caso, la base de datos que hemos ultizado se aloja en el HSS sobre mysql así que procedemos a la instalación en el servidor Asterisk de los drivers y librerias de mysql:

```
root@voice:~#apt-get install mysql-server  libmysqlclient-dev
```

Las tablas que consulta Asterisk no son las que ya existen en el HSS por lo que se ha creado una base de datos específica llamada asterisk con la estructura requerida. Esta base de datos contiene en realidad una vista que se alimenta de los datos ya existentes en el HSS relativos a los usuarios de la plataforma. El anexo C.7.1 contiene el detalle de creación de la base de datos, las tablas y las vistas.

También contiene la definición del usuario mysql que usará Asterisk, el cual en principio sólo debe de conectarse desde un único servidor:

```
GRANT ALL ON asterisk.* to asterisk@203.0.113.6 IDENTIFIED BY 'asterisk';
```

La primera vista, sip, generará la siguiente configuración para cada uno de los usuarios IMS:

```
name=alice
type=friend
secret=NULL
host=dynamic
callerid=alice@example.org
context=default
mailbox=alice
nat=no
qualify=no
fromuser=NULL
authuser=NULL
fromdomain=NULL
insecure=NULL
canreinvite=no
disallow=NULL
allow=NULL
restrictcid=NULL
ipaddr=NULL
port=NULL
regseconds=NULL
```

Donde los principales campos son:

- **name**: Usuario con el que nos conectaremos al servidor Asterisk.
- **type**: Define la clase de conexión que tendrá el cliente. Hay tres tipos de clientes SIP: peer (solo puede recibir llamadas), user (solo puede realizar llamadas) y friend (puede recibir y realizar llamadas).
- **secret**: Contraseña con la que se realizará la autenticación en Asterisk junto con el name. En nuestro caso, NULL indica que no requiere autenticación.
- **callerid**: Es el identificador del cliente, es decir, el nombre que aparecerá cuando se realice una llamada.
- **context**: Contexto asociado al cliente en el dialplan de Asterisk, es decir, contexto que se aplicará en el fichero extensions.conf del cual hablaremos más adelante.
- **host**: IP desde la que se permite el registro. El valor “dynamic” indica que puede ser cualquiera.
- **mailbox**: Nombre del buzón de voz asociado.
- **nat**: Si se establece a ‘yes’ activará métodos para descubrir se se haya detrás de un NAT y aplicar así soluciones de NAT traversal.
- **qualify**: Si se establece a ‘yes’ podremos monitorizar la extensión. Esta configuración no es muy recomendable cuando tenemos muchas extensiones.
- **canreinvite**: Si se establece a ‘yes’ se permitirá el reenvío de peticiones reINVITE. Se añadió este parámetro para poder deshabilitarlo por configuración como workaround antes dispositivos que no implementan correctamente este método.

Usando esta configuración, un teléfono puede realizar llamadas al servidor Asterisk utilizando el nombre de usuario `alice`, sin contraseña y con una dirección IP dinámica (host = dynamic). Dado que las llamadas siempre procederán de uno de los S-CSCF de la plataforma, hemos desactivado la funcionalidad de `nat`, la funcionalidad `qualify`, que genera peticiones SIP de tipo OPTIONS para determinar si el terminal se encuentra activo o no, y la funcionalidad `canreinvite` ya que asterisk será siempre el terminador de la llamada.

El campo `type` de una cuenta puede ser de tipo `peer` (otro servidor Asterisk por ejemplo) que permita realizar llamadas salientes des de el punto de vista del servidor Asterisk, o puede ser un usuario, tipo `user`, al que permitimos realizar llamadas entrantes (de nuevo des de la perspectiva del servidor).

El nuestro caso lo hemos definido de tipo `friend` que combina tanto user como peer.

El campo `context` determina la lógica de procesado que usará Asterisk al recibir una llamada desde o hacia ese usuario. Al igual que la configuración SIP se divide en diferentes cuentas, dialplan se divide en contextos.

Para procesar una llamada entrante, Asterisk requiere definir un contexto dentro del dialplan asociado a la cuenta que está realizando la llamada. Así que el contexto asociado en la configuración de la cuenta determina qué parte del dialplan es el que se debe buscar para una extensión marcada. La configuración del dialplan se encuentra en el fichero `extensions.conf`. Más adelante veremos como se define el contexto que utilizaremos, `default`.

La segunda vista, voicemail, creará la configuración del buzón de voz para cada abonado de IMS. En el caso de `alice` por ejemplo:

```
curtomer_id=alice
context=default
mailbox=alice
password=NULL
fullname=alice
email=alice@example.org
pager=NULL
STAMP=<fecha de creación>
```

Dado que el servidor Asterisk debe conectarse a la la base de datos Asterisk del HSS, hemos modificado el archivo de configuración general del demonio de mysql, anexo C.7.2, para que escuche peticiones a través de su IP pública.

Especificamos los datos de conexión para la base de datos en el fichero /etc/asterisk/res_mysql.conf:

```
[general]
dbhost = 203.0.113.5
dbname = asterisk
dbuser = asterisk
dbpass = asterisk
dbport = 3306
```

Especificamos en el fichero /etc/asterisk/extconfig.conf la priorización de mysql sobre los ficheros locales donde el servidor Asterisk debe buscar sus datos en tiempo real:

```
[settings]
sipusers => mysql,general,sip
sippeers => mysql,general,sip
voicemail => mysql,general,voicemail
queues => mysql,general,queues
queue_members => mysql,general,queue_members
```

El fichero de configuración /etc/asterisk/sip.conf, anexo C.7.3, contiene los parámetros generales de servicio. Los principales, IP y puerto de servicio, resolución de registros DNS de tipo SRV y dominios de servicio, se ven a continuación:

```
[general]
context=default         ; Contexto por defecto para las llamadas entrantes
allowguest=no           ; Denegamos llamadas des de orígenes anónimos
bindport=5060           ; Puerto UDP de servicio
bindaddr=0.0.0.0        ; IP de servicio
srvlookup=yes           ; Resolución de registros DNS de tipo SRV
domain=example.org      ; Dominio de servicio por defecto
domain=203.0.113.6      ; Dominios extra (local)
allowexternalinvites=no ; Desactivamos peticiones de tipo INVIE o REFER a otros dominios

#include sip_users.conf ; Incluímos un fichero específico para la definición de peers no alojados en la DB
```

El servidor recogerá la información de los usuarios de IMS de la base de datos del HSS pero necesitamos definir también Configuramos los S-CSCFs y el P-CSCF como peers en el fichero sip.conf para permitir que nos puedan enviar llamadas:

```
[pcscf]
type=friend
context=default
host=203.0.113.2
insecure=very

[scscf]
type=friend
context=default
host=203.0.113.3
insecure=very

[scscf2]
type=friend
context=default
host=203.0.113.9
insecure=very
```

Como hemos mencionado, la lógica de servicio se define en el fichero extensions.conf. En este caso sólo usaremos la funcionalidad de voicemail para cualquier destino dentro del contexto que hemos definido por defecto (default):

```
[default]
exten => _.,1,Voicemail(${EXTEN})
exten => _.,2,Hangup
```

Querremos también que los usuarios puedan consultar su buzón de voz. Para ello hemos de crear un usuario en la plataforma asociado a este propósito. En este caso sip:*86@example.org

Este usuario, que en realidad es ficticio, tendrá un comportamiento diferente por lo que hemos de definir en Asterisk un comportamiento específico para que presente el buzón de voz del llamante y no del llamado. Para ello, configuramos el extensions.conf tal que:

```
[default]
exten => _.,1,Voicemail(${EXTEN})
exten => _.,2,Hangup

exten => *86,1,VoicemailMain($CALLERIDNUM)
exten => *86,2,Hangup
```

Asterisk evalúa las rutinas que aparecen en el contexto seleccionado y ejecuta en primer orden las más específicas. En este caso, dado que esta nueva configuración aplica para una única extensión, se priorizará y será la seleccionada.

7.5. DNS

Con objeto de poder utilizar DDDS, es necesario configurar un servidor DNS que resuelva todos los registros necesarios para el descubrimiento de cada uno de los servicios.

En el escenario se ha configurado un servidor DNS implementado con Bind9 en la máquina virtual que hace las veces de encaminador.

El resto de máquinas se han configurado para que consulten este servidor DNS:

```
$ more /etc/resolv.conf

search example.org
options timeout:1 attempts:2
nameserver 203.0.113.1
```

Por su parte, en el servidor DNS se ha configurado como servidor autoritativo para el dominio example.org y el dominio inverso 113.0.203.in-addr.arpa:

```
$ more /etc/bind/named.conf.local
...
zone "113.0.203.in-addr.arpa" {
        type master;
        file "/etc/bind/db.203.0.113";
        allow-update {key "rndc-key";};

};
zone "example.org" {
        type master;
        file "/etc/bind/db.example.org";
        allow-update {key "rndc-key";};
};
```

El fichero /etc/bind/db.example.org contiene los registros necesarios para el autodescubrimiento de los servicios de la plataforma:

```
$ more /etc/bind/db.example.org

$ORIGIN example.org.
$TTL 1W
@                       1D IN SOA       localhost. root.localhost. (
                                        2006101001      ; serial
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum

                        1D IN NS        ns
ns                      1D IN A         203.0.113.1

pcscf                   1D IN A         203.0.113.2
_sip.pcscf              1D SRV 0 0 5060 pcscf
_sip._udp.pcscf         1D SRV 0 0 5060 pcscf
_sip._tcp.pcscf         1D SRV 0 0 5060 pcscf

icscf                   1D IN A         203.0.113.4
_sip                    1D SRV 0 0 5060 icscf
_sip._udp               1D SRV 0 0 5060 icscf
_sip._tcp               1D SRV 0 0 5060 icscf

example.org.       1D IN A         203.0.113.2
example.org.       1D IN NAPTR 10 50 "s" "SIP+D2U"      ""      _sip._udp
example.org.       1D IN NAPTR 20 50 "s" "SIP+D2T"      ""      _sip._tcp

scscf                   1D IN A         203.0.113.3
_sip.scscf              1D SRV 0 0 5060 scscf
_sip._udp.scscf         1D SRV 0 0 5060 scscf
_sip._tcp.scscf         1D SRV 0 0 5060 scscf

scscf2                   1D IN A         203.0.113.9
_sip.scscf              1D SRV 0 0 5060 scscf2
_sip._udp.scscf         1D SRV 0 0 5060 scscf2
_sip._tcp.scscf         1D SRV 0 0 5060 scscf2

hss                     1D IN A         203.0.113.5
presence                1D IN CNAME     scscf.example.org.
voicemail               1D IN A         203.0.113.6
```

7.6. Clientes SIP

Por último, con objeto de estudiar el comportamiento del escenario, se han evaluado diferentes clientes SIP.

A diferencia del entorno de estudio de un escenario SIP con *Kamailio* o *Asterisk*, en este escenario se ha descartado el cliente *Linphone* debido a su incapacidad para resolver el reto de validación que genera el S-CSCF.

Se valoró en su lugar el uso del cliente *Ekiga* pero se descartó también dado que el proceso de des-registro no se inicia ni al deshabilitar la cuenta de usuario ni al finalizar el programa.

Finalmente se decició prescindir de un cliente gráfico e instalar en el anfitrión el cliente *psjua*.

Configuración del cliente PJSUA

La opciones con las que se inician los clientea son:

```
# pjsua --id sip:usuario_ims@example.org  --registrar sip:example.org  --realm example.org \
--username usuario_ims@example.org --password pass_usuario_ims --proxy sip:pcscf.example.org \
--null-audio --auto-play --auto-answer 180 --play-file hello16000.wav \
--nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims --publish
```

Donde las opciones principales son:

- --id sip:*usuario_ims*@example.org: ID de usuario que se usará en el campo FROM de la cabecera SIP.
- --registrar sip:example.org: URI SIP del servicio de registro.
- --realm example.org: Dominio SIP local.
- --username *usuario_ims*@example.org: Usuario para el proceso de autenticación.
- --password *pass_usuario_ims*: Password de la cuenta de usuario.
- --proxy sip:pcscf.example.org: Configura el outbound proxy para cualquier petición SIP.
- --null-audio: Deshabilita los recursos locales de audio dado que no están disponibles en las máquinas virtuales.
- --auto-play –play-file hello16000.wav: Define el archivo de audio a ejecutar automáticamente al iniciarse el RTP de una llamada.
- --auto-answer 180: Código de respuesta automático al recibir un INVITE. En este caso se asocia con el estado RINGING.
- --nameserver=ns.example.org: Define el servidor DNS. En caso de no definirse, recoge el definido en el sistema operativo.
- --add-codec=PCMA/8000: Prioriza el codec PCMA sobre los demás. Nos interesa este codec para poder escuchar el audio determinadas versiones de Wireshark.
- --use-ims: Habilita las capacidades 3GPP/IMS del cliente, cabeceras **`PATH`** y **`Service-Route`**.
- --publish: Habilita el método PUBLISH para presencia.

7.7. Prácticas

7.7.1. Primeros pasos con IMS

En las siguientes prácticas se darán los primeros pasos utilizando el IMS. Para ello se analizará una configuración IMS incompleta utilizando el escenario ims-basic. La topología de este escenario se muestra en la Figura 7.10.

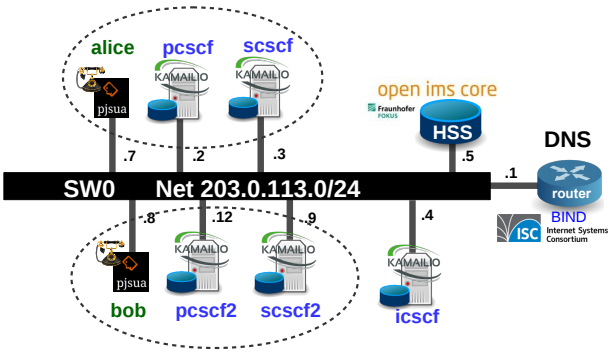


Figura 7.10: Esquema de red para las primeras pruebas de IMS básicas.

Como puede observar, la configuración a nivel de red es sencilla, con un solo switch conectando todos los nodos IMS. Los nodos IMS están implementados con Kamailio, el HSS es de open IMs core y el DNS es un típico `bind9` de ISC. Para arrancar el escenario teclee el siguiente comando en el hypervisor:

```
hypervisor$ simctl ims-basic start
```

Para realizar los siguientes ejercicios, configuraremos la traducción de nombres en `wireshark` para que sea más cómodo analizar los diferentes intercambios de mensajes. Para ello, introduzca el siguiente contenido en el fichero `$HOME/.config/wireshark/hosts` (o `$HOME/.wireshark/hosts` dependiendo de donde se almacene el perfil de `wireshark`):

```
203.0.113.1   dns
203.0.113.2   pcscf
203.0.113.3   scscf
203.0.113.4   icscf
203.0.113.5   hss
203.0.113.6   voiceas
203.0.113.7   alice
203.0.113.8   bob
203.0.113.9   scscf2
203.0.113.10  pras
203.0.113.11  claire
203.0.113.12  pcscf2
```

A continuación vaya al menu de `wireshark` Edit->Preferences->Name Resolution y marque las casillas *Only use the profile "hosts" file* y *Resolve network (IP) addresses* tal y como puede observar en la Figura 7.11. A continuación, capture en el interfaz `SimNet0` del **hypervisor**.

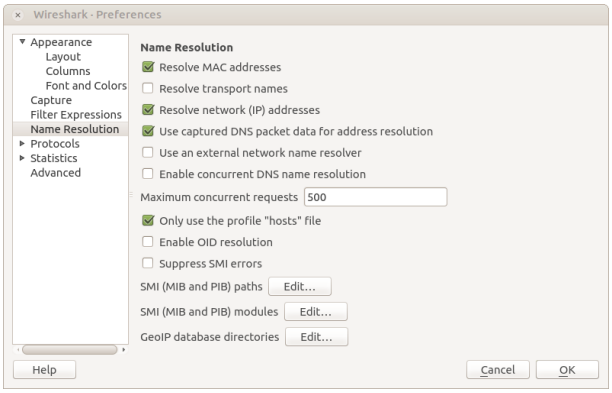


Figura 7.11: Activar un fichero hosts para resolución en Wireshark.

Ejercicio 7.1– En este ejercicio probaremos un intento de registro desde **alice**. Para ello, en un terminal en **alice** ejecute el siguiente comando para iniciar el registro:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims
```

Utilizando el tráfico capturado y un *Flow Graph* de `wireshark`, conteste a las siguientes cuestiones:

1. ¿Qué consultas al DNS realiza **alice** antes de iniciar el registro?
2. ¿Cuál es el primer elemento del IMS al que **alice** envía el REGISTER?
3. ¿Qué respuesta obtiene **alice**? ¿Cuál cree que es el motivo de que el registro falle (puede consultar el enunciado del siguiente ejercicio para obtener una pista)?

Ejercicio 7.2– En este ejercicio vamos a repetir el registro pero esta vez iniciamos el `pjsua` con un parámetro para indicar cual será el proxy a utilizar. Dicho parámetro es `--proxy sip:pcscf.example.org`. Por tanto, en **alice** tecleamos:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims \
--proxy sip:pcscf.example.org
```

Conteste a las siguientes cuestiones:

1. ¿Qué consultas DNS realiza **alice** antes de iniciar el registro?
2. ¿Cuál es el primer elemento del IMS al que **alice** envía el REGISTER?
3. ¿Cuál es el segundo? ¿Como lo identifica el P-CSCF?
4. ¿Qué consulta realiza el I-CSCF al HSS?
5. ¿Cuál es el motivo de que **alice** reciba un "403 - Forbidden - HSS Roaming not allowed" (puede consultar el enunciado del siguiente ejercicio para obtener una pista)?

7.7.2. Dominios

Para que el IMS funcione correctamente debemos configurar el dominio que sirve. En este caso, el dominio es example.org y lo debemos configurar en el HSS. El HSS se configura mediante un navegador y para ello vamos a ejecutar un `firefox` en el **hypervisor** utilizando un *namespace* propio. Ejecute los siguientes comandos:

```
hypervisor$ simtools-nsconf -a 203.0.113.11/25 -d 203.0.113.1 -s example.org
Creating a NS in the hypervisor...
Deleting any previous conf...
[sudo] password for testuser:
hypervisor# firefox &
```

En el navegador, conéctese al gestor de configuración del HSS utilizamos los siguientes datos:

`URL: https://203.0.113.5:8080`

`User: hssAdmin  Password: hss`

A continuación vamos mediante los menús de configuración del HSS a Network Configuration ->Visited Networks ->Search. Podremos observar que existe ninguna entrada. A continuación iremos a Network Configuration ->Visited Networks ->Create y crearemos la entrada example.org tal y como se muestra en la Figura 7.12.

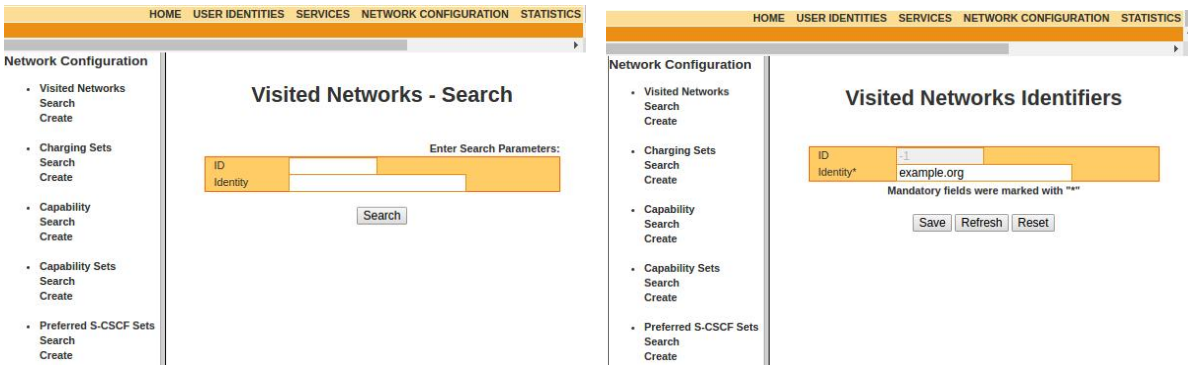


Figura 7.12: Menú HSS de creación de Visited Networks.

Nota importante: en <u>todos los menús del HSS</u> para realizar cambios, haga primero click en *Refresh*, luego realice los cambios y finalmente haga click en *Save* para que dichos cambios sean efectivos. Además, es aconsejable que siempre compruebe que sus cambios están cargados.

Ejercicio 7.3– En este ejercicio se trata de, una vez configurado el dominio en el HSS, repetir el registro de **alice** y analizar el resultado obtenido. Para ello capture en `SimNet0` e inicie el registro en **alice**:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file /root/hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims \
--proxy sip:pcscf.example.org
```

1. Analizando el tráfico capturado, ¿Qué respuesta se obtiene en este caso del HSS?

7.7.3. Servicios y Usuarios

Ahora vamos a crear los usuarios **alice** y **bob** pero antes debemos definir ciertos parámetros de servicio en el IMS.

S-CSCF y Service Profile

Para definir los S-CSCFs debemos ir a $Network\ Configuration \rightarrow Preferred\ S-CSCFSets \rightarrow Create$. En este menú podemos dar de alta los dos S-CSCF de nuestro entorno.

En particular, crearemos un S-CSCF Set donde el servidor prioritario será scscf.example.org y scscf2.example.org será el secundario. También crearemos otro set con los roles de los servidores invertidos. La Figura 7.13 ilustra la creación del primer set (llamado scscf.example.org).

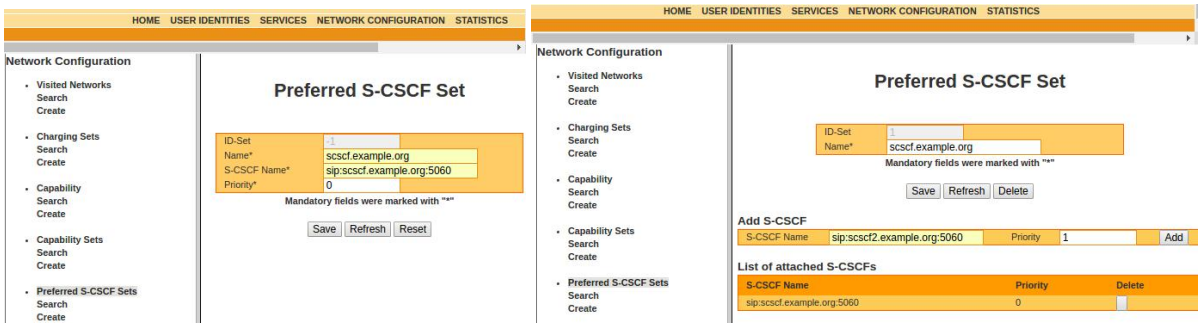
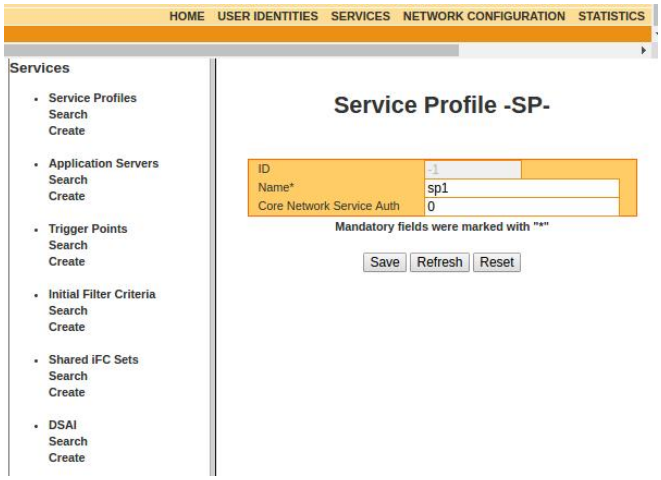


Figura 7.13: Menú HSS de definición de un *Preferred S-CSCF Set*.



Además, crearemos un service profile. Para realizar las prácticas, recuerde que debe crear un segundo set llamado scscf2.example.org con la prioridad de los S-CSCF invertida. Además, note que hemos creado un Service Profile (SP) llamado "sp1" mediante el menú:
Services ->Services Profiles ->Create
De momento este SP no contendrá ninguna directiva de servicio pero es obligatorio disponer de uno para la creación de los usuarios.

Alta de los abonados

Ahora ya podemos crear los abonados del IMS. Cada usuario de IMS (IMSU) dispone de al menos una identidad privada (IMPI) y una o más identidades públicas (IMPU). La identidad privada en IMS (IMPI) sigue la estructura NAI. Por ejemplo:

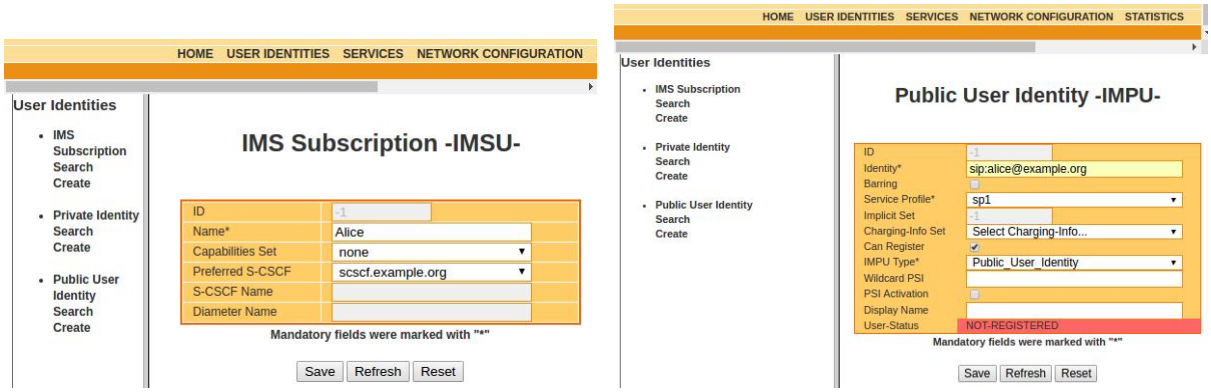
```
alice@example.org
```

La identidad de usuario pública (IMPU) utiliza el formato de SIP URI o TEL URI:

```
sip:alice@example.org
```

Para dar de alta los abonados empezaremos creando la subscripción al IMS. Para ello, utilizaremos el menú User Identities ->IMS Subscription ->Create y crearemos la suscripción de **alice** tal y como se muestra en la Figura 7.14a.

Al hacer click en `Refresh+Save` nos aparece una nueva pantalla en la que podemos crear y asociar un IMPI pero no lo hacemos en este momento (lo haremos un poco después). Recuerde que debe repetir el proceso para el usuario **bob**. Para este usuario seleccione scscf2.example.com para el campo "Preferred S-CSCF".



(a) IMSU (b) First sub-figure

Figura 7.14: Menús HSS de definición de IMSU/IMPU

Ahora daremos de alta la **Public User Identity (IMPU)** de los usuarios. Para ello, nos dirigimos al menú User Identities ->Public User Identity ->Create y creamos la identidad pública de **alice** según se muestra en la Figura 7.14b. Esta identidad la hemos creado de tipo `Public_User_identity`.

Una vez hacemos click en `Save`, nos aparece la pantalla que vemos en figura a continuación en la que debemos añadir el dominio bajo el cual **alice** podrá operar. Para ello, en el desplegable de Add Visited-Networks seleccione example.org y añádalo (ver Figura 7.15).

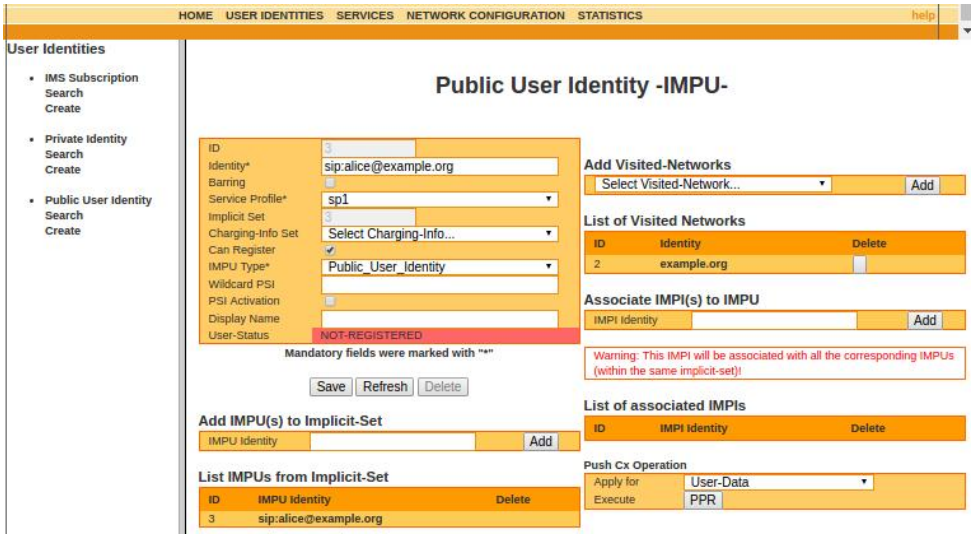


Figura 7.15: Menú HSS añadir *Visited Network*

.

Recuerde repetir el proceso para crear la identidad pública de **bob**.

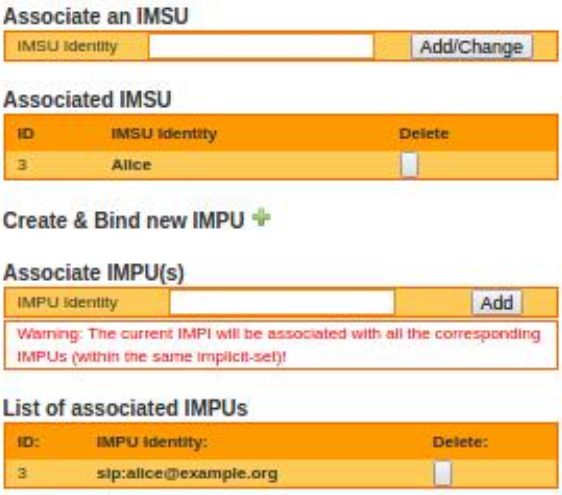
Ahora, vamos proceder a asociar la identidad privada o **Private User Identity (IMPI)** a los usuarios. Para ello, nos dirigimos a User Identities ->Private User Identity ->Create y creamos la identidad privada de **alice** según se muestra en la siguiente figura:

Los parámetros para el usuario **alice** son los siguientes:

- Identity: alice@example.org
- Secret Key: alice
- Authentication schemes: Seleccione la casilla `All`
- Default: Digest-MD5

Notar que hemos escogido como esquema de autenticación por defecto "Digest-MD5"debido a que es el que cliente `pjsua` implementa.

Una vez hacemos click sobre el botón `Save` nos aparecen nuevas opciones en las que haremos la asociación entre la identidad privada, la pública y la suscripción tal y como se muestra a continuación:



Nota. Asegúrese de que la configuración para **alice** queda tal y como muestra la figura y repita el proceso para crear la identidad privada de **bob** y asociarla a su identidad pública y a la suscripción.

Nota. Si ha realizado correctamente las configuraciones anteriores puede proceder con los siguientes ejercicios, pero, también puede ejecutar la etiqueta `users` para asegurarse que dispone de la configuración correcta.

Ejercicio 7.4– Vuelva a iniciar la captura con `wireshark` sobre `SimNet0` y realice de nuevo el registro del cliente `pjsua` de **alice**. Conteste a las siguientes cuestiones:

1. ¿Qué respuesta obtiene el I-CSCF en esta ocasión del HSS?
2. Tras el intercambio de mensajes entre el I-CSCF y el HSS, ¿cuál es siguiente elemento del IMS que interviene en el proceso del registro?
3. ¿Qué información dentro de la respuesta del HSS permite al I-CSCF conocer a los S-CSCF del escenario? Fíjese en el campo *Server-Capabilities*.
4. En el HSS mire el estado y los parámetros asignados tras el registro de **alice**.

Analizando el tráfico capturado mediante un *Flow Graph* vamos a hacer un análisis de las diferentes cabeceras de routing SIP durante el registro.

5. Respecto al REGISTER enviado por **alice**, ¿qué significado tiene la dirección en la *request line*? ¿por qué esta petición lleva un header Route? ¿cómo sabe **alice** qué valor debe tener este header?
6. ¿Qué nodo del IMS introduce la cabecera Path? ¿para qué sirve dicha cabecera? ¿por qué el I-CSCF no introduce una cabecera Path?
7. ¿Cuántos headers Via lleva el REGISTER cuando llega al S-CSCF correspondiente? ¿por qué?
8. ¿Por qué el 200 OK al registro no incluye cabeceras Route?
9. ¿Qué nodo del IMS introduce la cabecera Service-Route? ¿en qué mensaje SIP? ¿para qué sirve dicha cabecera?

Abra ahora la consola de **bob** y registre el cliente `pjsua` con las siguientes opciones:

```
bob# pjsua --id sip:bob@example.org  --registrar sip:example.org --realm example.org \
--username bob@example.org --password bob --null-audio --auto-loop --nameserver=ns.example.org \
--add-codec=PCMA/8000 --use-ims --proxy sip:pcscf2.example.org
```

10. ¿Cuál es el S-CSCF en el que se registra **bob**? ¿A qué se debe ese cambio?

Ejercicio 7.5– Vamos ahora a analizar una llamada entre los dos usuarios. Capture de nuevo el tráfico en `SimNet0` y realice una llamada de **bob** a **alice**:

```
>>>m
(You currently have 0 calls)
Buddy list:
 -none-

Choices:
   0          For current dialog.
  -1          All 0 buddies in buddy list
  [1 - 0]     Select from buddy list
  URL         An URL
  <Enter>     Empty input (or 'q') to cancel
Make call: sip:alice@example.org
```

Conteste a las siguientes cuestiones:

1. ¿Qué elementos del IMS y en qué orden intervienen en la transacción INVITE de la llamada? Utilice el análisis de llamadas de VoIP de `wireshark` y haga un *Flow Graph*. ¿Qué protocolos se utilizan entre cada par de nodos?
2. Describa y explique el por qué de todos los headers Via, Route y Record-Route de cada petición INVITE entre cada par de nodos SIP desde **bob** hasta **alice**.
3. ¿Qué consulta realiza el I-CSCF al HSS? ¿con qué propósito?
4. ¿Qué consulta realiza el I-CSCF al HSS? ¿con qué propósito?
5. Describa el trayecto SIP que siguen las peticiones ACK y BYE. ¿Por qué es distinto al INVITE?

7.7.4. Presencia en IMS

En este caso vamos a analizar el servicio de presencia SIP y su relación con el IMS. La configuración será la generada en los ejercicios anteriores pero además, se ha de crear un tercer usuario en el IMS llamado **claire**, con las siguientes características:

- IMSU: Claire
- P-CSCF: pcscf2.example.org
- Preferred S-CSCF: scscf2.example.org
- IMPU: claire@example.org
- IMPI: sip:claire@example.org
- Secret Key: claire

Presencia end-to-end

En primer lugar vamos a analizar el comportamiento de nuestro IMS ante la generación de los primeros métodos que se definieron dentro de la extensión de presencia para SIP: SUBSCRIBE y NOTIFY.

Ejercicio 7.6– Ejecute la etiqueta **`users`** que le sitúa en la configuración inicial requerida (con usuarios **alice**, **bob** y **claire** creados). Para ello, ejecute en el **hypervisor**:

```
hypervisor$ simctl ims-basic exec presence_ini
```

Iniciamos los clientes `pjsua` en **alice** y **bob** y los registramos en el IMS tal y como hemos visto en los ejercicios anteriores. A continuación, iniciamos la captura en `SimNet0`.

Ahora vamos a analizar los eventos de presencia. Tenga en cuenta que nuestro escenario no dispone de ningún AS específico para presencia, por lo que son los clientes los que deben procesar y contestar a dichas peticiones. Ahora, cree el *buddy* de **alice** en **bob** con el siguiente comando en el `pjsua` de **bob**:

```
>>>+b
Enter buddy's URI: (empty to cancel): sip:alice@example.org
```

Tras la creación del *buddy*, conteste a las siguientes cuestiones:

1. ¿Qué tipo de petición genera **bob**?
2. ¿Qué elemento del escenario responde a la petición? ¿observa alguna diferencia en el encaminamiento por el IMS respecto a la petición INVITE?
3. ¿Qué otro método de SIP aparece? ¿Quién lo genera y quién es el destinatario? ¿cuál es su objetivo? Para ello fíjese en el cuerpo de la petición y en la cabecera Subscription-State de dicha petición.

A continuación vamos a ejecutar un `pjsua` en el **hypervisor** para que se registre como el usuario **claire** en el IMS **utilizando el terminal en el *namespace* que ya tenemos lanzado**:

```
hypervisor# pjsua --id sip:claire@example.org  --registrar sip:example.org --realm example.org \
--username claire@example.org --password claire --auto-answer 180 --proxy sip:pcscf2.example.org \
--nameserver=203.0.113.1  --add-codec=PCMA/8000 --use-ims --ip-addr=203.0.113.11 \
--contact=sip:claire@203.0.113.11
```

Veamos ahora cual es el proceder de un cliente cuando hay más de un suscriptor. Vuelva a capturar en `SimNet0` y cree en **claire** el *buddy* de **alice**. A continuación, modifique el estado del cliente `pjsua` en **alice** a "offline" utilizando:

```
>>>T
```

Y luego:

```
>>>7
```

Analizando el tráfico capturado, conteste a las siguientes cuestiones:

4. ¿Cuantas peticiones de tipo NOTIFY genera **alice**? ¿Quién son los destinatarios?
5. ¿En qué parte de la petición **alice** informa de su nuevo estado?

Presencia con SIMPLE

Ahora se van a analizar los eventos de presencia usando el protocolo SIMPLE. En SIMPLE un servidor es el encargado de gestionar los eventos de presencia. Además, en este esquema aparece un nuevo método SIP denominado PUBLISH. Por tanto, en este caso, se utilizará un AS de presencia SIMPLE disponible en el escenario denominado **pras** (ver figura 7.16).

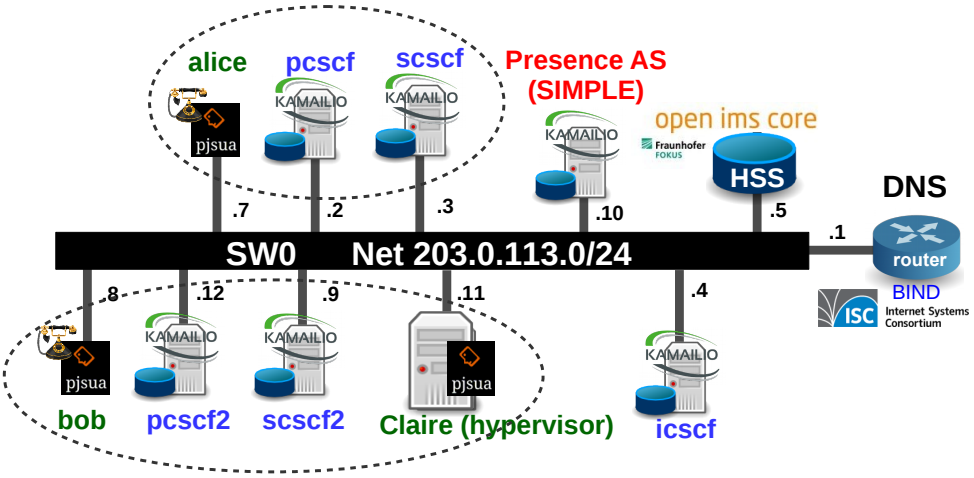


Figura 7.16: IMS con servidor SIMPLE de presencia.

Ejercicio 7.7– En primer lugar, iniciamos la captura de tráfico en `SimNet0` e iniciamos el cliente `pjsua` de **alice** indicando que utilice el método PUBLISH. Para ello, en **alice**:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims \
--proxy sip:pcscf.example.org --publish
```

Analizando el tráfico capturado, conteste a las siguientes cuestiones:

1. ¿Qué otro método de SIP aparece después del proceso de registro?
2. ¿Qué respuesta obtiene **alice** para este nuevo método? ¿Qué elemento del escenario lo genera? Fijándonos en el campo **`to`** de las cabeceras de la petición, ¿cuál es la razón de este comportamiento?

Configuración del AS de Presencia

Si deseamos gestionar la presencia de un modo centralizado y más cómodo necesitaremos desplegar un AS dedicado a ello en nuestro IMS. En nuestro escenario disponemos de un servidor de presencia llamado **pras**. Este sevidor de presencia está implementado con un kamailio que tiene las extensiones y módulos de presencia habilitados.

Sin embargo, este servidor se encuentra aislado ya que la base de datos del HSS no tiene ninguna información al respecto. A continuación, conectaremos el servidor **pras** al IMS y analizaremos el nuevo comportamiento del escenario ante eventos de presencia. Para conectar un AS con el IMS, debemos configurar una serie de elementos en el HSS:

a) En primer lugar, nos conectamos con un navegador web al HSS como ya hemos visto y creamos el nuevo Application Server. Esto se realiza en la sección *Services ->Application Servers ->Create* tal y como se muestra en la Figura 7.17.

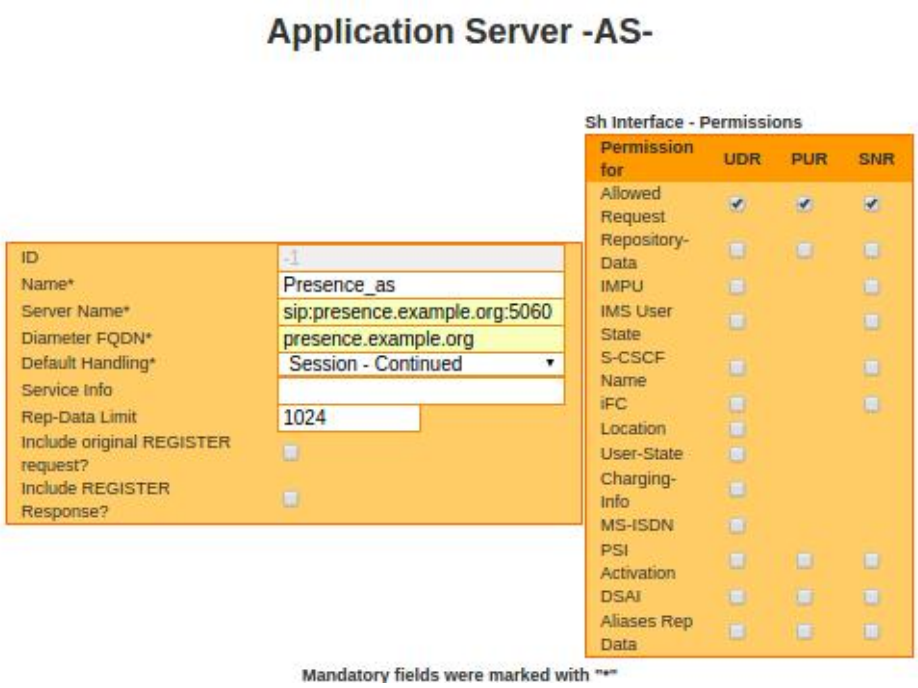


Figura 7.17: Menú HSS de creación de un AS.

b) Ahora el HSS conoce la existencia del nuevo AS pero desconoce bajo que condiciones se le deben enviar peticiones. Estas condiciones se definen en los *Trigger Points*. Nos dirigimos a la sección *Services ->Trigger Points ->Create* y configuramos un nuevo *Trigger Point* como se muestra en la Figura 7.18.

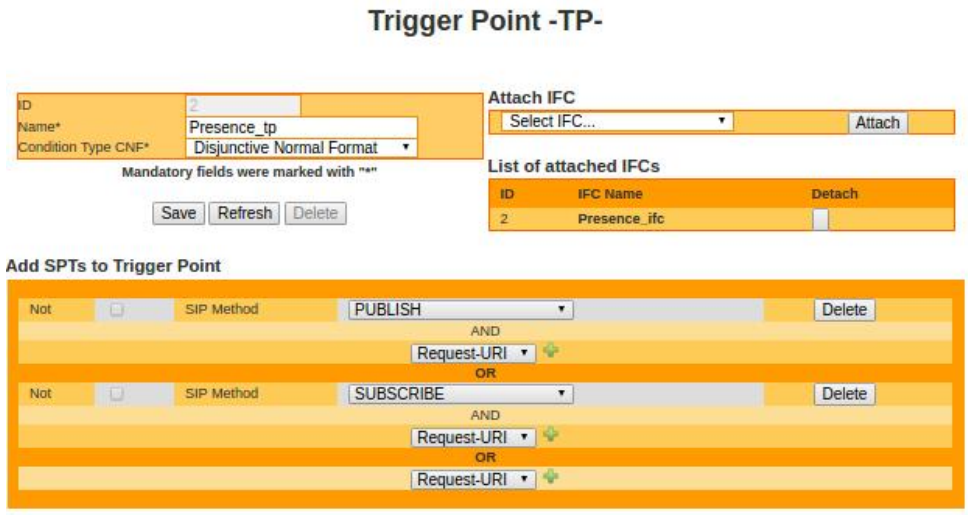


Figura 7.18: Menú HSS de creación de un *Trigger Point*.

c) En este punto tenemos un AS y un trigger point definido, pero no están relacionados entre ellos. Para ello necesitamos crear un *Initial Filter Criteria*. Nos dirigimos a la sección *Services ->Initial Filter Criteria ->Create* y configuramos la asociación de los dos elementos como se muestra en la Figura 7.19.

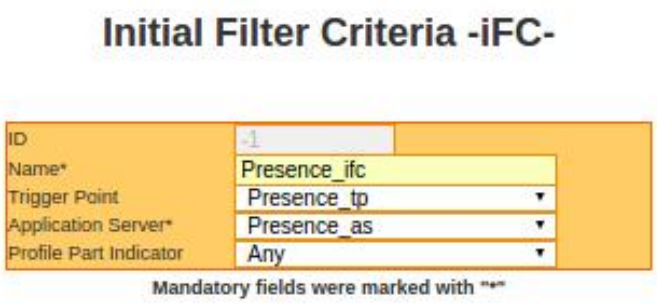


Figura 7.19: Menú HSS de creación de un *Initial Filter Criteria*.

d) En IMS, los usuarios disponen de un perfil de servicio que es en el que se especifican los servicios adicionales disponibles. En nuestro escenario, todos los usuarios tienen asociado el perfil *sp1* que en realidad no contiene nada. Vamos ahora a crear un nuevo perfil de servicio que contendrá el de presencia. Para ello nos dirigimos a la sección *Services ->Service Profile ->Create* y creamos el nuevo perfil *ims_services* en el que asociaremos el iFC *Presence_ifc*, tal y como se muestra en la Figura 7.20.

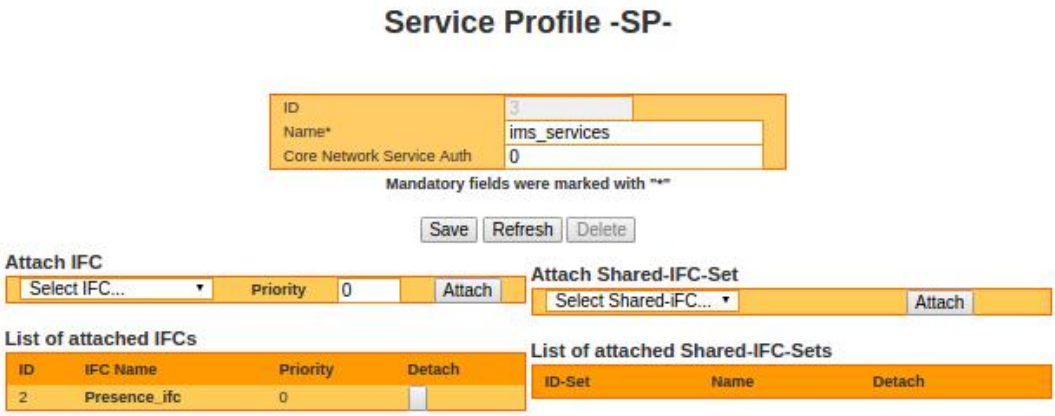


Figura 7.20: Menú HSS de definición de un *Service Profile*.

e) Por último, modificaremos el perfil de servicio asociado a **alice**, **bob** y **claire** en la definición de sus respectivas identidades públicas como se muestra en la Figura 7.21.

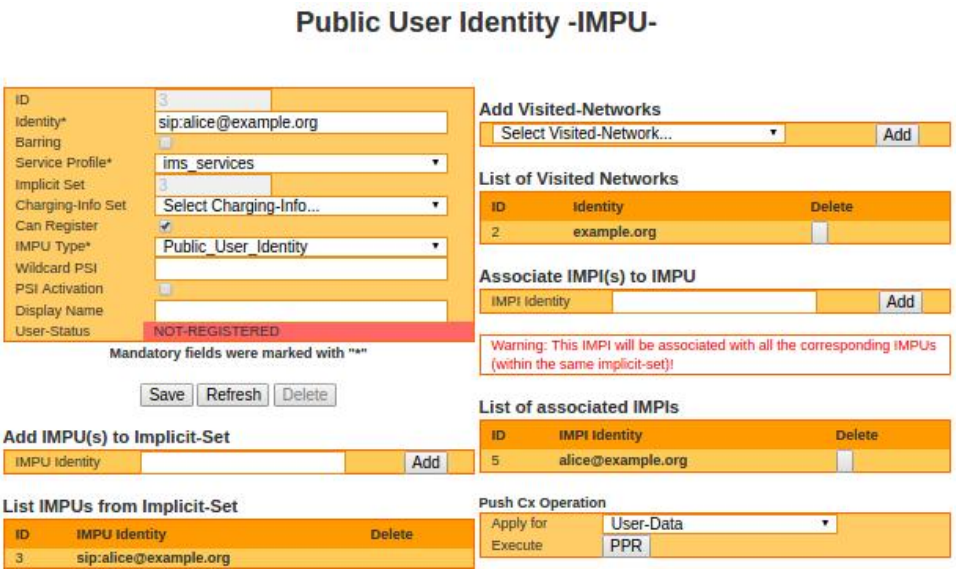


Figura 7.21: Menú HSS de configuración de IMPUs con el AS de presencia.

Siempre que hagamos un cambio en algún elemento de un perfil de servicio, deberemos **forzar el envío de las nuevas características a los S-CSCF**. Esto se puede conseguir:

(I) Desde el cliente SIP con un re-registro de usuario.

(II) Desde el HSS, dentro de la configuración de la identidad pública de los usuarios, forzando un *Push Cx Operation*.

En cualquier caso, antes de ejecutar el push haciendo click sobre el botón *PPR*, habilitaremos el debug de la consola del HSS para que nos muestre la información enviada. Este debug se activar haciendo click en la opción *"TURN ON - DEBUG"* del apartado *STATISTICS*.

Ejercicio 7.8– Una vez realizada la configuración anterior o bien ejecutando la etiqueta **`presence`**, capturamos el tráfico de la `SimNet0` y volvemos a la configuración de las identidades públicas de los usuarios donde forzamos el "*push*" de los datos.

Nota. Para evitar ver el tráfico intercambiando mediante el protocolo HTTP con el HSS, en `wireshark` aplicaremos el siguiente filtro: `!(tcp.port == 8080)`

Conteste a las siguientes cuestiones:

1. Si volvemos al apartado de *STATISTICS*, ¿qué nueva información aparece en la consola de logging?
2. Fijándonos en la captura de tráfico, ¿qué protocolo es el utilizado para esta comunicación?
3. ¿Cuál es el servidor destino de la actualización para cada usuario?

Ejercicio 7.9– Ahora que ya tenemos un AS de presencia incorporado al escenario, vamos a analizar su comportamiento. Para ello, cerramos todos los clientes `pjsua` (tecla **q**), capturamos en `SimNet0` e iniciamos el cliente `pjsua` de **alice** para que se registre de nuevo en el IMS utilizando también la opción `--publish`:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims \
--proxy sip:pcscf.example.org --publish
```

Conteste a las siguientes cuestiones:

1. ¿Qué elementos del escenario intervienen en la petición PUBLISH generada por **alice**?
2. ¿Qué elemento genera la respuesta en esta ocasión? ¿Que código de respuesta es el generado?

Ahora iniciamos los clientes `pjsua` de **bob** y **claire**, ambos con la opción `--publish`. También reiniciamos la captura en `SimNet0` y creamos el *buddy* correspondiente a **alice** en **bob** y en **claire**. Sin parar la captura de tráfico, cambiamos el estado de presencia en **alice** a offline. Utilizando el tráfico capturado, conteste a las siguientes cuestiones:

3. ¿Qué elemento del escenario responde a la peticiones de tipo SUBSCRIBE de **bob** y **claire**?
4. ¿Qué elemento del escenario informa a **bob** y a **claire** del estado de presencia de **alice**?
5. ¿Cree que **alice** tiene algún conocimiento de sus suscriptores? ¿Por qué?
6. ¿Cuantas peticiones SIP se han generado a raíz del cambio de estado de **alice**? ¿Qué elemento/elementos las han generado?

7.7.5. Buzón de voz

En este último apartado crearemos otro servicio en nuestro IMS. En este caso será para el buzón de voz de los usuarios. El objetivo es que las llamadas se desvíen al buzón de voz bajo las siguientes condiciones:

- El usuario final no se encuentra registrado.
- El usuario final no responde en un determinado intervalo de tiempo (en la práctica utilizaremos un valor de 8 segundos).
- El usuario origen desea consultar su buzón de voz.

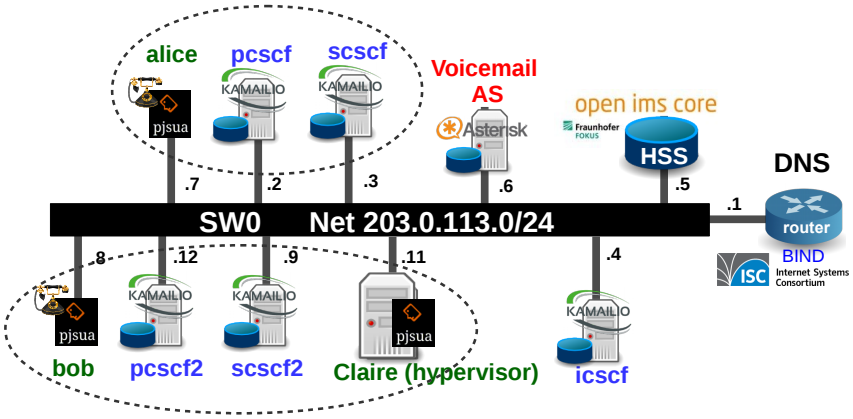


Figura 7.22: Esquema de red para las primeras pruebas de IMS básicas.

Como se muestra en la Figura 7.22, el escenario dispone de un AS para las tareas de buzón de voz denominado **voice**, en el que se ha instalado una PBX Asterisk. En el HSS, se ha creado una vista especial de mysql que será la que consulte Asterisk para la integración del buzón de voz con la base de datos centralizada de usuarios.

Aunque el servidor de buzón de voz ya se encuentra configurado, deberá realizar una serie de modificaciones de configuración con el fin de interconectar el IMS con el nuevo servicio de buzón de voz.

Nota. Como punto de partida ejecute la etiqueta `presence` en `simctl` para ajustar el escenario a la configuración inicial necesaria para este ejercicio.

Desvío al buzón de voz para usuarios finales no registrados

De forma análoga a como se hizo para el servicio de presencia, modificaremos el Service Profile **ims_services** para incluir un nuevo iFC relativo al voicemail:

1. Datos para la configuración del AS **voicemail**:

```
Name: Voicemail_as
Server Name: sip:voicemail.example.org:5060
Diameter FQDN: voicemail.example.org
Default Handling: Session - Continued
```

2. Datos para la configuración del *Trigger Point*:

```
Name: Voicemail_tp
Condition Type CNF: Disjunctive Normal Format
SPTs:
        Session Case:    Term - UnReg
                            AND
        Method:                  INVITE
```

3. Datos para la configuración del iFC:

```
Name: Voicemail_ifc
Trigger Point: Voicemail_tp
Application Server: Voicemail_as
Profile Part Indicator: Any
```

Incluya el iFC que ha creado en el service profile **ims_services** con prioridad **1**.

Nota. La configuración anterior también se puede conseguir ejecutando la etiqueta `voicemail_ifc`.

Ejercicio 7.10– Para iniciar el ejercicio, salga o cancele todos los clientes `pjsua`, de esta forma todos los usuarios quedarán sin registro en el IMS. A continuación, registre con un `pjsua` a **alice** en el IMS:

```
alice# pjsua --id sip:alice@example.org  --registrar sip:example.org --realm example.org \
--username alice@example.org --password alice --null-audio --auto-play --auto-answer 180 \
--play-file hello16000.wav --nameserver=ns.example.org  --add-codec=PCMA/8000 --use-ims \
--proxy sip:pcscf.example.org
```

1. Consultando en el menú de identidades públicas del HSS a través del navegador, ¿qué estado aparece para cada uno de los usuarios?

Ahora, inicie una captura en `SimNet0` y a continuación, realice una llamada desde **alice** a **claire** (en la dirección sip:claire@example.org), espere 10 segundos antes de terminar dicha llamada. Analizando el tráfico capturado, conteste a las siguientes cuestiones:

2. ¿Qué información incluye la petición inicial que realiza el I-CSCF al HSS? ¿qué información retorna como respuesta el HSS?
3. Analizando la captura explique el intercambio de mensajes DIAMETER que ha tenido lugar. ¿qué estado aparece ahora para cada uno de los usuarios? ¿a qué se pueden deber cada uno de estos estados?

   Para responder a las preguntas anteriores consulte los menús del HSS de identidades públicas y también los suscripciones IMS (IMSU). Como pista, debe tener en cuenta que el I-CSCF tiene una base de datos local con la información de todos los S-CSCF disponibles en el IMS. Si el I-CSCF no obtiene del HSS la localización de un usuario, puede enviar una determinada petición SIP al S-CSCF que considere oportuno.
4. Vuelva a capturar el tráfico en `SimNet0` y realice otra llamada de **alice** a **claire**, ¿qué elementos del escenario intercambian mensajes DIAMETER? ¿qué información observamos esta vez?

Iniciamos ahora el cliente `pjsua` en el *namespace* creado en el **hypervisor** para registrar a **claire** en el IMS:

```
hypervisor# pjsua --id sip:claire@example.org  --registrar sip:example.org --realm example.org \
--username claire@example.org --password claire --auto-answer 180 --proxy sip:pcscf2.example.org \
--nameserver=203.0.113.1  --add-codec=PCMA/8000 --use-ims --ip-addr=203.0.113.11 \
--contact=sip:claire@203.0.113.11
```

Analizando el tráfico capturado, conteste a la siguiente cuestión:

5. ¿En qué S-CSCF se registra **claire**? ¿es el que hemos configurado como preferente? ¿cuál puede ser la causa?

Desvío al buzón de voz por tiempo máximo de espera

Los *Trigger Points* son una herramienta muy útil en el IMS pero no sirven para controlar el comportamiento de nuestro escenario ante cualquier situación. Un ejemplo de ello es el caso en el que queremos que la llamada se desvíe al buzón de voz si el usuario destino no contesta en X segundos. Es por ello que en ciertos casos, también es necesario modificar directamente la lógica de procesado de los proxies (kamailio en nuestro caso).

Ejercicio 7.11– Antes de modificar dicha configuración, vamos a realizar una llamada de **alice** a **claire** (ambos están registrados) sin contestarla y esperaremos a que la llamada sea cancelada. Analizando el tráfico capturado, conteste a las siguientes cuestiones:

1. ¿Qué nodo del escenario cancela la llamada?¿al cabo de cuanto tiempo?
2. ¿Cuál es el código de respuesta?

Ejercicio 7.12– Vamos ahora a modificar el comportamiento de los dos S-CSCF para que las llamadas sean desviadas al buzón de voz en caso de que el tiempo de espera supere los 8 segundos o que el S-CSCF reciba un código de estado de error. Antes de ello, será necesario **des-registrar todos los clientes** `pjsua` para evitar fallos de inconsistencia en las bases de datos.

Un vez des-registrados los clientes, introduzca los siguientes dos cambios en el fichero de configuración de kamailio de los dos S-CSCF del IMS del escenario:

(I) Reenviar hacia el voice mail cuando se produzcan errores en el método INVITE:

Nos conectamos a la consola de **scscf.example.org** y editamos el fichero /etc/kamailio/kamailio.cfg. Escribiremos el código siguiente entre las líneas 930 y 932.

```
#LAB - Inicio aplicar ruta error Voicemail para INVITE
if (is_method("INVITE")) {
        #Introducimos la ruta al Voicemail en caso de error
        if(!t_is_set("failure_route")) t_on_failure("VOICEMAIL");
        if (impu_registered("location")) {
                #Modificamos el timeout de llamada a 8 segundos
                #para usuarios registrados en la plataforma
               t_set_fr(8000);
        }
}
#Fi LAB - Fin aplicar ruta error Voicemail para INVITE
```

Las líneas anteriores tienen el texto de referencia "**#LAB - Inicio aplicar ruta error Voicemail para INVITE**" y "**#Fi LAB - Fin aplicar ruta error Voicemail para INVITE**" para su fácil localización.

(II) Configuración de la propia ruta de error hacia el voicemail. Para ello, nos dirigimos al final del fichero /etc/kamailio/kamailio.cfg y añadimos el siguiente código:

```
#LAB - Ruta al Voicemail
failure_route[VOICEMAIL] {
        xlog("L_ERR","User unavailable, voicemail route forced\n\n");
        if (impu_registered("location")) {
                rewritehostport("voicemail.example.org:5060");
                t_relay();
        }
}
#Fi LAB
```

Las líneas anteriores también se encuentran marcadas en el fichero de configuración de Kamailio.

Finalmente, aplicamos la nueva configuración:

```
scscf# service kamailio restart
```

A continuación debe repetir los pasos anteriores para el segundo S-CSCF (**scscf2.example.org**).

NOTA: El escenario dispone también de la etiqueta `voicemail_route` que al ejecutarla aplicará los cambios de configuración en los dos S-CSCF.

Una vez realizada la configuración anterior, volvemos a registrar los clientes, capturamos el tráfico de `SimNet0` y realizamos una llamada de **alice** a **claire** como en el ejercicio anterior. Analizando el tráfico capturado, conteste a las siguientes cuestiones:

1. ¿Qué elemento del escenario contesta la llamada?
2. ¿Qué elemento del escenario desvía la llamada al buzón de voz? ¿Al cabo de cuanto tiempo?

Ejercicio 7.13– Las modificaciones que hemos introducido no sólo afectan al tiempo de espera antes de desviar la llamada al buzón de voz, si no que este desvío también se producirá en caso que el S-CSCF reciba un código de error como respuesta al INVITE. Para probar esta circunstancia, una forma sencilla de generar un código de error es rechazar la llamada. Veamos como se comporta el escenario ante este caso.

Capturamos el tráfico de `SimNet0` y realizamos una llamada de **alice** a **claire**. Esta vez, en **claire** rechazaremos la llamada pulsando la tecla **h**. Analizando el tráfico capturado, conteste a las siguientes cuestiones:

1. ¿Qué elemento del escenario contesta la llamada?
2. ¿Qué elemento del escenario desvía la llamada al buzón de voz? ¿Cuando?

Consultas al buzón de voz

Por último, debemos permitir que los usuarios puedan consultar su buzón de voz. Para ello, configuraremos una nueva condición en el *Trigger Point* **Voicemail_tp**.

Ejercicio 7.14– Implemente en el HSS la siguiente configuración:

```
Name: Voicemail_tp
Condition Type CNF: Disjunctive Normal Format
SPTs:
        (Session Case:  Term - UnReg) AND (Method:      INVITE)
        OR
        (Method:        INVITE) AND (Request URI:       sip:*86@example.org)
```

Nota. También puede situarse en la configuración requerida en este punto ejecutando la etiqueta `complete`.

1. Con esta nueva condición, ¿cuál debería ser el flujo de proxies que atraviesa la llamada?

Capturamos el tráfico en `SimNet0`, registramos el cliente `pjsua` de **claire** y llamamos a **sip:*86@example.org**. Analizando el tráfico capturado, conteste a las siguientes cuestiones:

2. ¿Cuál es efectivamente el flujo de proxies que atraviesa la llamada? ¿coincide con la predicción de la pregunta anterior?

Capítulo 8

Conclusiones

El objetivo principal de este Proyecto Final de Carrera era la implementación de un sistema IMS que dispusiera de las funciones S-CSCF, P-CSCF, I-CSCF, HSS y dos AS.

Para llegar a este objetivo, primero se desarrolló un estudio de SIP y se consolidaron los conocimientos teóricos mediante la implementación y estudio de los siguientes escenarios:

- **Llamada directa entre UAs:** Análisis de las cabeceras y de la negociación de RTP y RTCP a través de SDP. Inicialmente mediante el uso del cliente `pjsua`, se amplió este escenario a la incorporación de otro cliente, `linphone`.
- **Registro en un servidor registrar y llamada a través de proxy SIP (Triángulo)** Se incorpora un servidor *Kamailio* introduciendo el concepto de suscriptor, descubrimiento dinámico por DNS y el proceso de REGISTRO. En el siguiente escenario, realiza las funciones de servidor registrar y inbound proxy SIP. Se analiza una llamada a través de éste haciendo énfasis en las cabeceras SIP que parecen por primera vez: **`Record-Route`**, **`Via`** y **`Route`**.
- **Llamada a través de inbound y outbound proxy (Trapezoide)** Se añade un segundo proxy para completar el esquema del trapezoide.
- **Llamada directa entre UAs detrás de NAT y a través de proxy SIP** Primera visión de como afecta NAT a SIP y la negociación del RTP. A continuación, se analizan los puntos fuertes y debilidades de la solución que ofrece un proxy SIP.
- **Llamada a través de proxy SIP con apoyo de STUN:** Se analizan los puntos fuertes y debilidades del protocolo STUN.
- **Llamada a través de proxy SIP con apoyo de TURN:** Se analizan los puntos fuertes y debilidades del protocolo TURN.
- **Llamada a través de proxy SIP con apoyo de ICE:** Se analiza el proceso de negociación ICE.
- **Llamada a través de proxy SIP con apoyo del módulo NAThelper de Kamailio y RTPproxy:** Se analizan los puntos fuertes y debilidades de la solución y, en concreto, la interacción con ICE.
- **Llamada directa entre UAs mediante el uso de SRTP y SIP inseguro:** Se analizan las debilidades de la solución y, para mayor ilustración, se descifra el flujo media.
- **Llamada directa entre UAs mediante el uso de SRTP y SIP sobre TLS:** Se analiza la solución y se descifra la comunicación mediante la clave privada generada para TLS.
- **Seguridad de una llamada entre UAs a través de proxy SIP:** Se analiza y se exponen los riesgos de la seguridad de SRTP y SIP sobre TLS cuando hay diferentes saltos. Se analiza la solución de SIPS y el cifrado extremo a extremo.

A continuación se realizó el estudio teórico de IMS y se desarrolló la implementación del sistema, inicialmente compuesto por un S-CSCF, un P-CSCF, un I-CSCF, todos ellos con *Kamailio*, un HSS basado en `FHoSS` y dos servidores de aplicaciones, `Kamailio` y `Asterisk`.

Las prácticas desarrolladas sobre este escenario están focalizadas en el análisis del sistema, no específicamente de SIP. El estudio se realiza de una forma más amplia considerando aspectos como la interacción entre los diferentes nodos, roles y la aparición de otro protocolo *DIAMETER*.

Durante el desarrollo de las mismas, se vio la necesidad de incluir nuevos nodos que permitieran dotar de una visión más didáctica al escenario. Para ello se incorporó:

- **Segundo S-CSCF:** Permitió analizar de forma mas amigable el comportamiento del sistema frente a una llamada entre usuarios al disponer cada uno de ellos un serving propio. Además, se analizó el proceso de selección de S-CSCF que realiza el I-CSCF en caso de no obtener este dato del HSS.
- **Segundo P-CSCF:** Al tener un único proxy, se hacía complicado en algunos casos determinar si el proxy actuaba como proxy del abonado A o del abonado B. Se ve la necesidad de añadir un segundo proxy que permita separar los abonados entre ambos con el objetivo de analizar de forma más clara el camino de los mensajes SIP.

Una vez se dispuso de un sistema funcional con las funciones principales, se amplió el escenario con dos servicios, presencia y buzón de voz. Ello permitió configurar el sistema con condiciones que modificaran el comportamiento de IMS y encaminara los mensajes SIP a los nuevos servidores.

Incorporamos otra aplicación altamente extendida en el mundo VoIP, `Asterisk`, configurado para leer de la base de datos del HSS la información de los abonados de la plataforma. Para ello hubo que realizar una vista mysql específica en modo lectura en la base de datos del HSS que solo el servidor `Asterisk` puede consultar y que presenta la información relativa a los abonados en el formato que `Asterisk` espera.

Así, al finalizar el proyecto, no sólo se han cubierto los objetivos iniciales sino que se ha desarrollado una plataforma IMS con una gran diversidad de proxies. Además, se han ido desarrollando puntos de restauración entre el final e inicio de las prácticas que permiten configurar el escenario de forma rápida y recuperarlo frente a errores.

Cabe mencionar que no se ha incluido en la memoria la resolución de las prácticas dado que éstas se utilizan como material docente.

8.1. Líneas de futuro

IMS es una arquitectura basta y compleja lo que abre gran cantidad de posibles líneas de futuro. A continuación se exponen las cuatro lineas de futuro que yo desarrollaría inicialmente:

- **Bases de datos centralizadas:** El S-CSCF, P-CSCF y I-CSCF disponen, en este proyecto, de una base de datos local. En el caso del serving y el proxy es especialmente interesante externalizar esta base de datos ya que estas se alimentan en tiempo real. Dicha externalización permitirá capturar la interacción entre `Kamailio` y `Mysql` en cada uno de los nodos. Por ejemplo, en el caso del S-CSCF tendríamos las consultas a la base de datos de localización de los usuarios,, los procesos de registro y des-registro y las consultas a los iFC al recibir una petición. En el caso del serving y el proxy es especialmente interesante externalizar esta base de datos ya que estas se alimentan en tiempo real. Dicha externalización permitirá capturar la interacción entre *Kamailio* y *Mysql* en cada uno de los nodos.
- **WebRTC:** `Kamailio` dispone de un módulo de soporte para websockets. Otra línea de futuro consistiría en añadir otro AS para la integración de WebRTC en esta plataforma. los procesos de registro y des-registro y las consultas a los iFC al recibir una petición.
- **NAT en IMS:** Los dos P-CSCF tienen los módulos de apoyo a NAT configurados y el servicio de RTPproxy. Dado que no se han hecho pruebas, no se menciona esta configuración en la memoria pero si deja una vía abierta a ampliar las prácticas con ejercicios de IMS y NAT.
- **Interacción entre plataformas IMS:** Sería muy interesante levantar dos plataformas como las desarrolladas en este proyecto y analizar la interacción entre sistemas autónomos IMS. Incluso añadiendo previamente otros elementos IMS como SBC o BGRF.
- **WebRTC:** *Kamailio* dispone de un módulo de soporte para *Websockets*. Otra línea de futuro consistiría en añadir otro AS para la integración de WebRTC en esta plataforma.

Apéndice A

Kamailio

Kamailio (anteriormente llamado SER y OpenSER) es un servidor SIP de código abierto bajo licencia GPL que implementa todas las entidades lógicas conocidas en un entorno VoIP: servidor proxy, redirect, registrar y location.

Galardonado como el mejor del software de código abierto de redes 2009 por la revista InfoWorld, es ampliamente utilizado en todo el mundo sirviendo a millones de abonados y miles de millones de sesiones enrutadas al mes.

Gracias a su arquitectura modular permite adaptar el servicio a las necesidades de cada plataforma cargando sólo aquellos módulos necesarios como contabilidad, autenticación, gestión de NAT, soporte HTTP, XML-RPC, así como WebSockets (para el apoyo WebRTC).

Es un proyecto que nace de la disgregación de SER en verano del 2005 en dos equipos de trabajo independientes. Originalmente llamado openSER, en Julio de 2008, debido a problemas de derechos de autor pasa a llamarse kamailio y, en Noviembre de ese mismo año los dos proyectos vuelven a unificarse bajo este mismo nombre. La integración finaliza con la versión 3.0.0 en la que se unen los dos códigos en uno solo.

A.1. Arquitectura

Kamailio posee una arquitectura modular, lo que le permite ser altamente flexible y funcional en cualquier tipo de escenario. Esta arquitectura está compuesta básicamente por dos partes principales:

- **Núcleo:** Encargado de las funcionalidades de bajo nivel necesarias para los módulos como gestión de memoria, DNS, gestor de la capa de transporte, retransmisión sin estado (call stateless), SIP Message parser, etc. Desde la versión 3.0 también contiene las llamadas "internal libraries" que se componen de código compartido por gran cantidad de módulos pero que no tienen un propósito general para ser parte del núcleo principal.
- **Módulos:** Son los componentes que proporcionan las distintas funcionalidades y que lo hacen tan poderoso. Actualmente hay más de 150 módulos diponibles como gestión de registro y localización, autorización y autenticación, procesado de transacciones SIP con estado, presencia, IM, conectores de bases de dados SQL y no-SQL, módulos específicos para IMS, etc.

A.2. Procesado de mensajes SIP

El fichero de configuración de Kamailio no sólo contiene los parámetros que esperaríamos encontrar en un fichero de configuración como direcciones IPs y puertos de servicio, carga de módulos, parámetros de conexión a bases de datos, número de procesos, etc, si no que también contiene la lógica del procesado de los mensajes SIP. Es decir, cuales son los pasos a realizar cuando se recibe o se envía un mensaje SIP.

Este procesado que se lleva a cabo en la recepción de un mensaje varía en función de si se trata de una petición o una respuesta.

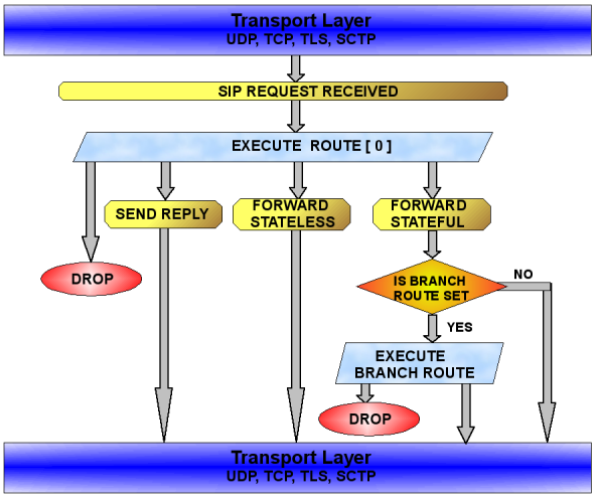


Figura A.1: Lógica de procesado para peticiones SIP.

Como muestra la figura A.1, al recibir una petición SIP ésta es siempre procesada en primer lugar en la route[0] o ruta principal. En ella se pueden establecer distintas gestiones para la petición:

- Si la petición no cumple las condiciones de seguridad establecidas se descarta (drop).
- Si el destino de la petición es kamailio, genera una respuesta.
- Si el destino no es kamailio, es decir, es necesario retransmitirla, ejecuta la acción de retransmisión con o sin estado según lo indique el modo de configuración.

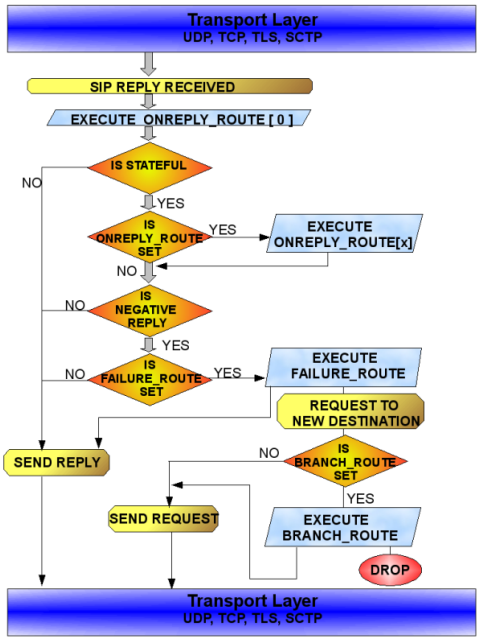


Figura A.2: Lógica de procesado para respuestas SIP.

En cambio, si el mensaje recibido es una respuesta, la lógica de procesado seguida es la mostrada en la figura A.2. Al recibir la respuesta, se ejecuta en primer lugar la ruta onreply_route[0]. El procesado depende de si el modo de trabajo es stateless o statefull:

- Si el modo es stateless, simplemente se retransmite la respuesta al siguiente salto.
- si el modo es statefull, se continuará con el procesado. En dicho procesado podremos especificar cómo actuar en caso de que se trate de un código de error.

A.3. Configuración

Por lo general, toda la configuración de Kamailio se lleva a cabo en el archivo **kamailio.cfg** bajo el directorio base que hayamos elegido para realizar la instalación, habitualmente "/etc/kamailio".

La configuración se divide en tres grandes bloques:

- **Configuración global:** Contiene la definición de las variables generales que afectan al núcleo como la dirección IP, puerto, nivel de log, y directivas que determinan que parte del código se ejecutará dando como resultado un menor consumo de memoria y mayor rapidez en la ejecución. Además, permite definir y cargar otro fichero que contenga parámetros de configuración lo que facilita la gestión de clústers de servicio en los que el fichero kamailio.cfg es común a todos y un fichero local contiene los parámetros específicos de cada nodo.
- **Sección de módulos:** Se indican los módulos que se van a utilizar a través del comando "loadmodule".
- **Configuración de módulos:** Contiene la definición de los parámetros específicos de los módulos cargados en la sección anterior. Esta definición se hace mediante el paso de parámetros con la función "modparam(nombre_módulo, parámetro_módulo, valor_parámetro)".
- **Lógica de enrutamiento:** Este bloque contiene la lógica de procesado de los mensajes SIP. A su vez se divide en diferentes secciones:
  - **Rutina principal:** A partir de este punto finaliza la definición de parámetros u opciones y comienza la lógica de procesado. Esta sección contiene la lógica principal.
  - **Rutinas secundarias:** Son opcionales. Kamailio permite definir rutas secundarias que serán invocadas desde la principal. El objetivo de estas rutas es evitar que la principal se convierta en una rutina demasiado grande y compleja.
  - **Rutinas en caso de error:** Son opcionales. Permiten la ejecución de rutinas específicas para el control de errores producidos al no procesarse de la forma esperada las anteriores rutinas o en la recepción de un código de error en una respuesta SIP.

Veamos este fichero de configuración a través de un ejemplo. Para ello, analizaremos los puntos más relevantes del fichero de configuración de nuestro nodo "scscf.example.org".

```
####### Global Parameters #########
include_file "scscf.cfg" [1]

#!ifdef WITH_DEBUG [2]
debug=5
log_stderror=no
sip_warning=yes
#!else
debug=2
log_stderror=no
sip_warning=no
#!endif

user_agent_header="User-Agent: Kamailio S-CSCF"
server_header="Server: Kamailio S-CSCF"

# LAB - Activamos naptr, default no
dns_try_naptr=yes [3]

loadmodule "tm.so" [4]
loadmodule "rr.so" [5]
loadmodule "dialog_ng.so" [6]

# -- usrloc params --
modparam("ims_registrar_scscf", "max_contacts", 5); [7]
#!ifdef DB_URL [8]
#!ifdef DB_URL2
modparam("ims_usrloc_scscf", "db_url", "cluster://cluster1")
#!else
modparam("ims_usrloc_scscf", "db_url", DB_URL)
#!endif
modparam("ims_usrloc_scscf", "db_mode", 1)
#!endif
modparam("ims_registrar_scscf", "default_expires",  604800) [9]
modparam("ims_registrar_scscf", "min_expires", 3600)
modparam("ims_registrar_scscf", "max_expires",  604800)

# -- CDP params --
modparam("cdp","config_file","/etc/kamailio/scscf.xml") [10]
```

[1] Indica que hay que añadir el contenido del fichero "scscf.cfg" a la configuración.

[2] Permite activar o desactivar el nivel de DEBUG en función de si se ha definido la variable "WITH_DEBUG".

[3] Activa el uso de resolución de registros DNS de tipo NAPTR para descubrimiento dinámico.

[4] Habilita el módulo para el procesado de transacciones en modo statefull.

[5] Activa el módulo que contiene la lógica de `Record-Route`.

[6] Módulo para las funcionalidades "Call statefull".

[7] Configura el parámetro de máximo número de contactos (localizaciones) registrados por usuario a 5.

[8] Configura el método de acceso a la base de datos en función de si se han definido las variables "DB_URL" y "DB_URL2".

[9] Configura diferentes temporizadores para el tiempo de vida de un registro.

[10] Configura el parámetro del fichero de configuración del módulo cdp. Este módulo es el encargado de las comunicaciones mediante el protocolo DIAMETER.

Como hemos comentado, parte de la configuración puede residir en otro fichero. Este fichero se suele utilizar, por ejemplo, para separar la configuración común en un clúster de servicio y la configuración particular de cada nodo. En el desarrollo de este proyecto hemos seguido esa filosofía. A continuación vemos un extracto del fichero "scscf.cfg" del nodo scscf.example.org:

```
# IP-Adress for incoming SIP-Traffic, in the following format:
listen=udp:203.0.113.3:5060 [1]
listen=tcp:203.0.113.3:5060

alias=scscf.example.org:5060 [2]

#!define NETWORKNAME "example.org" [3]
#!define NETWORKNAME_ESC "example\.org"
#!define HOSTNAME "scscf.example.org"
#!define URI "sip:scscf.example.org:5060"
#!define HOSTNAME_ESC "scscf\.example\.org"

# ENUM-Server to query:  [4]
#!define ENUM_SUFFIX "e164.arpa."

# Connection URL for the database:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/kamailio" [5]

# Select Authorization Algorhithm:
# Let the HSS decide
#!define REG_AUTH_DEFAULT_ALG "HSS-Selected" [6]
```

[1] Define la dirección IP, puerto y protocolo de transporte del servidor. Como vemos pueden existir varios.

[2] Es un identificador de servicio que utiliza tanto para construir cabeceras como para identificarse a sí mismo.

[3] Parámetros del dominio de servicio, nombre del servidor y uri SIP de identificación del servidor.

[4] Define el dominio para resoluciones de enum.

[5] Parámetros de conexión a la base de datos. Pueden definirse hasta dos, en ese caso, el segundo parámetro es DB_URL2.

[6] Indicamos que el algoritmo de validación debe ser el que indique el HSS.

Veamos ahora un ejemplo de las rutinas de procesado de mensajes SIP de nuestro scscf (fichero /etc/kamailio/kamailio.cfg). Empezamos por la principal, "route[0]" o "route":

```
route {  [1]
#!ifdef WITH_DEBUG  [2]
        xlog("L_ERR", "$rm ($fu ($si:$sp) to $tu, $ci)\n");
#!endif

        # per request initial checks
        route(REQINIT);  [3]

        # Handle Registrations:
        if (is_method("REGISTER")) {  [4]
                route(REGISTER);
                exit;
        }
```

```
        # we need to support subscription to reg event
        if (is_method("SUBSCRIBE") && search("^(Event|o)([ \t]*):([ \t]*)reg")) {
                route(SUBSCRIBE);
                break;
        }

        if (is_method("PUBLISH") && search("^(Event|o)([ \t]*):([ \t]*)reg")) {
                route(PUBLISH);
                break;
        }

        #Set DLG flag to track dialogs using dialog2  [5]
        if (!is_method("REGISTER|SUBSCRIBE"))
                setflag(FLT_DIALOG);

        # Evaluate Route-Header and set $route_uri
        loose_route();

        if (is_method("CANCEL|ACK")) {  [6]
                t_relay();
                exit;
        }

        if (($route_uri =~ "sip:orig@"+HOSTNAME_ESC+".*") || isc_from_as("orig")) {  [7]
                # we need something like this to assign SCSCF to unregistered user for services
                # support for AS origination on behalf of unregistered user
                # can use the registrar is_registered methods - must see if we need to check orig or term?

                # Originating
                route(orig);
                break;
        } else {
                isc_from_as("term");
                if ($retcode == -2) {
                        # Treat as originating, since it was retargeted:
                        route(orig);
                        break;
                }
                if ((is_in_profile("orig") || has_totag()) && ($route_uri =~ "sip:mo@"+".*")) {
                        route(orig_subsequent);
                        break;
                }
                if ((is_in_profile("term") || has_totag()) && ($route_uri =~ "sip:mt@"+".*")) {
                        route(term_subsequent);
                        break;
                }

                # Terminating
                if (uri=~"sip:(.*)@"+NETWORKNAME_ESC +"(.*)" || uri=~"tel:.*") {
                        if (!term_impu_registered("location")) {
                                xlog("L_DBG", "We need to do an UNREG server SAR assignemnt");
                                assign_server_unreg("UNREG_SAR_REPLY", "location", "term");
                                exit;
                        }
                } else {
                        sl_send_reply("403","Forbidden - Dialog not found on S-CSCF or \
                        Terminating user not suitable for unregistered services");
                        exit();
                }
                route(term);  [8]
                break;
        }
}
```

[1] Definición de la rutina “route”.

[2] Dentro de las rutinas también podemos activar o desactivar código en función de si se han definido ciertas variables. En este caso, si definimos “WITH_DEBUG” se invoca a la función que escribe en el log, xlog para que deje la siguientre entrada con serveridad de errir (“L_ERR”):

```
<request_method> (<from> (<ip_origen_mensaje_sip>:<puerto_origen_mensaje_sip>) to <to>, <call-id>)
```

[3] Invoca a una rutina secundaria con nombre REQINIT.

[4] Invoca a la rutina secundaria REGISTER siempre y cuando el método recibido sea un REGISTER.

[5] Activa un flag para el mantenimiento de estado del diálogo en caso que el método no sea ni REGISTER ni SUBSCRIBE.

[6] Hace una retransmisión directa en caso de que el método sea un CANCEL o un ACL y a continuación finaliza la rutina principal.

[7] Si la URI de la primera cabecera Route es similar a la expresión regular o si la petición proviene de un AS (isc_from_as), invoca a la rutina "orig":

- La expresión regular "sip:orig@-HOSTNAME_ESC+".*" tiene su sentido en IMS. En el proceso de registro, el S-CSCF envía la cabecera "Service-Route" inicializada a "sip:orig@<alias_scscf>" para indicar que debe estar en el camino de todas las peticiones que origine el usuario y que se realicen mientras dure el registro. De esta forma además, el S-CSCF puede identificar que es un llamada desde un usuario de la plataforma ya que esta cabecera la eliminará el S-CSCF una vez haya procesado la petición.
- La función "isc_from_as" pertenece al módulo ISC encargado de la implementación de interfaces ISC con servidores de aplicaciones SIP. En este escenario, el S-CSCF no se comunica con ningún AS mediante ese protocolo por lo que esta función siempre retorna negativo.

[8] **[8]** Invoca a la rutina secundaria "term". Esta es la rutina que se ejecuta cuando el destino de la llamada es un usuario registrado en el S-CSCF. A continuación veremos esta rutina ya que hemos hecho modificaciones en ella.

```
route[term] [1]
{
        xlog("L_DBG","Term\n");

        set_dlg_profile("term");

        #we need something like this to check if a user is barred
        # if (S_terminating_barred()){ [2]
                # sl_send_reply("404","Not Found - Terminating user barred");
                # exit;
        # }

        if (is_method("INVITE|SUBSCRIBE")) { [3]
                $avp(RR_CUSTOM_USER_AVP)="mt";
                $avp(i:20)="mt";
                record_route();
        }

        # check if dialog saved as fwded to AS [4]
        if (isc_match_filter("term","location")){
                t_on_failure("isc_term_failure");
                xlog("L_DBG","Term - msg was fwded to AS\n");
                exit;
        }

        if (lookup("location")) { [5]
                if (uri=~"sip:(.*)@"+NETWORKNAME_ESC+"(.*)") {
                        if (!t_newtran()) {
                                sl_reply_error();
                                exit;
                        }
                        t_reply("404","Not Found - destination user not found on this S-CSCF");
                        exit;
                }
        } else {
                # User not registered? Reply with 404.
                if (!t_newtran()) {
                        sl_reply_error();
                        exit;
                }
                t_reply("404","Not Found - destination user not found on this S-CSCF");
                exit;
        }
        route(apply_privacy);

        #LAB - Inicio aplicar ruta error Voicemail para INVITE
        if (is_method("INVITE")) { [6]
                #Introducimos la ruta al Voicemail en caso de error
                if(!t_is_set("failure_route")) t_on_failure("VOICEMAIL");
                if (impu_registered("location")) {
                        #Modificamos el timeout de llamada a 8 segundos
                        #para usuarios registrados en la plataforma
                       t_set_fr(8000);
                }
        }
        #Fi LAB - Fin aplicar ruta error Voicemail para INVITE

        if (!t_relay()) {
                sl_reply_error();
        }

        t_relay();
}
```

[1] Definición de la rutina "term".

[2] Si al usuario de le aplicado alguna política de bloqueo, no se le permiten recibir llamadas. En nuestro escenario no se han contemplado.

[3] Si el método es un INVITE o un SUBSCRIBE, añade una cabecera **`Record-Route`** para indicar que debe estar en el camino de sucesivas peticiones.

[4] La función "isc_match_filter" mira si existe algún iFC para el usuario cuando es destino de una petición, si existe, lo ejecuta y finaliza la rutina. En esta parte también se define la rutina de error "isc_term_failure". La activación de este tipo de rutinas se hace a través de la función "t_on_failure".

[5] En este caso comprobamos las condiciones que indicarán que el usuario no está registrado en este S-CSCF y como la rutina es la de llamadas terminales, hay que enviar como respuesta un "404 - NOT FOUND". Además se puede observar que hay una comprobación de si se puede generar o no la respuesta 404 (sino se enviaría un error interno mediante la función `s1_reply_error`). Las condiciones citadas son:

- Si la información de localización tiene la forma: *username*@*ourdomain* esto significa que el usuario **no está correctamente registrado** en este S-CSCF ya que en caso de estar correctamente registrado su información de registro debería tener la forma de la dirección de un P-CSCF.
- Si directamente no encontramos información de localización también significa que el usuario no está registrado en este S-CSCF.

[6] Esta es la rutina que hemos modificado para desviar las peticiones de tipo INVITE al voicemail en caso de error. Básicamente:

- *is_method("INVITE")* retorna positivo si la petición es un INVITE
- *t_on_failure("VOICEMAIL")* establece como rutina de error la llamada VOICEMAIL.
- *impu_registered("location")* devuelve positivo si la IMPU de la cabecera **`TO`** está registrado en este servidor.
- *t_set_fr(8000)* establece 8 segundos como *timeout* de espera de una respuesta final cuando el método es INVITE. Si se vence el *timeout* entraríamos a la rutina de error VOICEMAIL.

Es decir, si recibimos un INVITE dirigido a un usuario que tenemos registrado, configuramos como *timeout* en espera de respuesta final 8 segundos (el *timeout* general se ha dejado a 10 segundos que es la configuración que viene por defecto). Además, si se produce algún error al procesar el mensaje o si recibimos un código de error como respuesta final, procesaremos la rutina VOICEMAIL que vemos a continuación.

```
#LAB - Ruta al Voicemail
failure_route[VOICEMAIL] { [1]
        xlog("L_ERR","User unavailable, voicemail route forced\n\n"); [2]
        if (impu_registered("location")) { [3]
                rewritehostport("voicemail.example.org:5060");  [4]
                t_relay(); [5]
        }
}
#Fi LAB
```

[1] Definición de la rutina "VOICEMAIL". Notar que esta vez hay que especificar que se trata de una ruta en caso de error o "failure_route". Si no se especifica de esta forma, la rutina no se ejecutará ya que no hará coincidencia.

[2] Invocamos al módulo que gestiona la escritura en el log para dejar una entrada.

[3] La función *impu_registered("location")*, como ya hemos visto, devuelve positivo si la IMPU del **`TO`** está registrado en este servidor.

[4] La función *rewritehostport("<ip_o_fqdn>:<puerto>")* modifica el dominio de la **`Request-URI`** por los parámetros indicados. En este caso, apuntamos al nombre FQDN y puerto de servicio del servidor de voicemail.

[5] La función *t_relay()* retransmite el mensaje SIP ya procesado.

Por último, Kamailio, en la configuración que se distribuye con la distribución no contiene el código de adaptación de cualquier número a formato internacional. Eso hace que ENUM y NRENUM no sean consultables. A continuación se muestran las modificaciones realizadas en los proxy SIP de las prácticas relacionadas para normalizar la numeración a formato internacional:

```
# Check for Enum destinations
# if you want to make ENUM work with numbers starting with "00",
# use the following to convert "00" it into a "+"

if (uri=~"^tel:00[1-9][0-9]*") { [1]
        # strip leading "00"
        # (change example.net to your domainname or skip the stuff after the "@")
        strip(2); [2]
        # (adjust, if your international prefix is something else than "00")
        prefix("+"); [3]
};
# Enum works with international numbers so we must prepend "+"
if (uri=~"^tel:[1-9][0-9]*") {
        prefix("+"); [4]
};
# check if request uri starts with an international phone
# number (+X.), if yes, try to ENUM resolve in e164.arpa.
# if no result, try in nrenum.net
if (uri=~"tel:\+[0-9]") { [5]
        if ( !enum_query("e164.arpa.") ) { [6]
                enum_query("nrenum.net."); [7]
        };
};
```

[1] Compara la uri de la cabecera **TO** con la expresión regular que siguen los números de telefonía convencional con código internacional en formato “00XX”.

[2] Si la comparación anterior es positiva, elimina los dos primeros dígitos.

[3] Añade el carácter “+” al inicio siguiendo así el formato internacional.

[4] Añade el prefijo “+” en cualquier otro caso también ya que es necesario.

[5] Hacemos una última comprobación del formato de las uris tel ya que si no son correctos, las consultas enum provocarán un error.

[6] La función “enum_query” lanza una petición a DNS para resolver la uri tel bajo el dominio enum indicado, en este caso “e164.arpa.”.

[7] En caso de no hayar resolución en ENUM, probamos en NRENUM.

Apéndice B

Referencia comandos MYSQL

En este anexo se expone una pequeña referencia a consultas mysql útiles para ver la interacción con las bases de datos. Dado que, a excepción de la del AS de voicemail todas se encuentran en local, la captura de wireshark no permite ver esta interacción:

1. Conectarse a la base de dados en los X-CSCF:

```
# mysql -u root -p kamailio
Enter password: <enter: no hay password>
```

2. Conectarse a la base de dados en el hss:

```
# mysql -u root -p hss_db
Enter password: <enter: no hay password>
```

3. Consultar las tablas disponibles:

```
mysql> show tables
```

4. Consultar el número de filas por tabla en los X-CSCF:

```
mysql> SELECT TABLE_NAME, TABLE_ROWS FROM `information_schema`.`tables`  WHERE  \
`table_schema` = 'kamailio';
```

5. Consultar el número de filas por tabla en el hss:

```
mysql> SELECT TABLE_NAME, TABLE_ROWS FROM `information_schema`.`tables`WHERE  \
`table_schema` = 'hss_db';
```

6. Consultar en un S-CSCF la localización de los usuarios:

```
mysql> select * from contact;
```

7. Consultar en un S-CSCF las identidades públicas registradas:

```
mysql> select * from impu;
```

8. Monitorización con watch para ejecutar una consulta cada segundo y poder ver el desfase entre la eliminación del la localicación y el registro de las identidades públicas registradas en los S-CSCF:

```
# watch -n 1 mysql -u root -p -h localhost kamailio -Bse "source /tmp/watch_users.sql"´
```

9. Salir de base de dados:

```
mysql> exit
```

Apéndice C

Escenario IMS: Ficheros de configuración

C.1. Archivos de configuración generales

C.1.1. Escenario VNUML: ims_basic.vnuml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">

<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>ims-basic</simulation_name>
    <automac/>
    <vm_mgmt type="none" />
    <vm_defaults exec_mode="mconsole">
       <filesystem type="cow">/usr/share/vnuml/filesystems/debian6.fs</filesystem>
       <kernel>/usr/share/vnuml/kernels/linux-3.3.8</kernel>
       <!-- <console id="0">xterm</console> -->
    </vm_defaults>
  </global>
<net name="Net0" mode="uml_switch" hub="yes" sock="/var/run/vnuml/Net0.ctl"/>
<!--net name="Net1" mode="vde_switch" hub="yes"  sock="/var/run/vnuml/Net0.ctl/ctl" /-->
  <!-- SimNet0 config : sudo ifconfig SimNet0 203.0.113.1/24/-->

  <vm name="router">
    <console id="0">pts</console>
    <console id="1">pts</console>
    <if id="1" net="Net0">
        <ipv4>203.0.113.1/24</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/bind/" seq="start">./files/ims-basic/bind/</filetree>
    <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
    <exec seq="start" type="verbatim">/etc/init.d/bind9 restart</exec>
  </vm>

  <vm name="pcscf">
    <mem>256M</mem>
    <console id="0">pts</console>
    <console id="1">pts</console>
    <if id="1" net="Net0">
        <ipv4>203.0.113.2/24</ipv4>
    </if>
    <route type="ipv4" gw="203.0.113.1">default</route>
    <filetree root="/etc/kamailio" seq="start">./files/ims-basic/pcscf/etc.kamailio/</filetree>
    <filetree root="/etc/default" seq="start">./files/ims-basic/pcscf/etc.default/</filetree>
    <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
    <filetree root="/tmp" seq="start">./files/ims-basic/pcscf/tmp</filetree>
    <exec seq="start" type="verbatim">/etc/init.d/mysql start  &amp;amp;&amp;amp; /tmp/kam_db_create.sh \
    &amp;amp;&amp;amp; /tmp/kam_pcscf_create.sh</exec>
    <exec seq="start" type="verbatim">/etc/init.d/rtpproxy start</exec>
    <exec seq="start" type="verbatim">/etc/init.d/kamailio start</exec>
    <exec seq="initial" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
    <exec seq="users" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
    <exec seq="presence" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
```

```
  <exec seq="voicemail_ifc" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="complete" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
</vm>

<vm name="pcscf2">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.12/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/kamailio" seq="start">./files/ims-basic/pcscf2/etc.kamailio/</filetree>
  <filetree root="/etc/default" seq="start">./files/ims-basic/pcscf2/etc.default/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/pcscf2/tmp</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql start  &amp;&amp; \
  /tmp/kam_db_create.sh &amp;&amp; /tmp/kam_pcscf_create.sh</exec>
  <exec seq="start" type="verbatim">/etc/init.d/rtpproxy start</exec>
  <exec seq="start" type="verbatim">/etc/init.d/kamailio start</exec>
  <exec seq="initial" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="users" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="presence" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="voicemail_ifc" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
  <exec seq="complete" type="verbatim">/tmp/kam_pcscf_restore.sh</exec>
</vm>

<vm name="scscf">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.3/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/kamailio" seq="start">./files/ims-basic/scscf/etc.kamailio/</filetree>
  <filetree root="/etc/default" seq="start">./files/ims-basic/scscf/etc.default/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/scscf/tmp</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql start  &amp;&amp; /tmp/kam_db_create.sh \
  &amp;&amp; /tmp/kam_scscf_create.sh</exec>
  <exec seq="start" type="verbatim">/etc/init.d/kamailio restart</exec>
  <exec seq="initial" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="initial" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="users" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="users" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="presence" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="presence" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="voicemail_ifc" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="voicemail_ifc" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="voicemail_route" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.voicemail \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="complete" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.voicemail \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="complete" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
</vm>

<vm name="scscf2">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.9/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/kamailio" seq="start">./files/ims-basic/scscf2/etc.kamailio/</filetree>
  <filetree root="/etc/default" seq="start">./files/ims-basic/scscf2/etc.default/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/scscf2/tmp</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql start &amp;&amp; /tmp/kam_db_create.sh \
  &amp;&amp; /tmp/kam_scscf_create.sh</exec>
  <exec seq="start" type="verbatim">/etc/init.d/kamailio restart</exec>
  <exec seq="initial" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="initial" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="users" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="users" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="presence" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
```

```
  <exec seq="presence" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="voicemail_ifc" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.orig \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="voicemail_ifc" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="voicemail_route" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.voicemail \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
  <exec seq="complete" type="verbatim">/bin/cp -p /etc/kamailio/kamailio.cfg.voicemail \
  /etc/kamailio/kamailio.cfg &amp;&amp; /etc/init.d/kamailio restart</exec>
  <exec seq="complete" type="verbatim">/tmp/kam_scscf_restore.sh</exec>
</vm>

<vm name="pras">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.10/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/kamailio" seq="start">./files/ims-basic/pras/etc.kamailio/</filetree>
  <filetree root="/etc/default" seq="start">./files/ims-basic/pras/etc.default/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/pras/tmp</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql start  &amp;&amp; /tmp/kam_db_create.sh</exec>
  <exec seq="start" type="verbatim">/etc/init.d/kamailio restart</exec>
  <exec seq="initial" type="verbatim">/tmp/kam_pras_restore.sh</exec>
  <exec seq="users" type="verbatim">/tmp/kam_pras_restore.sh</exec>
  <exec seq="presence" type="verbatim">/tmp/kam_pras_restore.sh</exec>
  <exec seq="voicemail_ifc" type="verbatim">/tmp/kam_pras_restore.sh</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/kam_pras_restore.sh</exec>
  <exec seq="complete" type="verbatim">/tmp/kam_pras_restore.sh</exec>
</vm>

<vm name="icscf">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.4/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/kamailio" seq="start">./files/ims-basic/icscf/etc.kamailio/</filetree>
  <filetree root="/etc/default" seq="start">./files/ims-basic/icscf/etc.default/</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/icscf/tmp/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql start  &amp;&amp; /tmp/kam_icscf_create.sh</exec>
  <exec seq="start" type="verbatim">/etc/init.d/kamailio restart</exec>
</vm>

<vm name="hss">
  <mem>128M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.5/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/init.d" seq="start">./files/ims-basic/hss/etc.init.d</filetree>
  <filetree root="/etc/mysql" seq="start">./files/ims-basic/hss/etc.mysql</filetree>
  <filetree root="/tmp" seq="start">./files/ims-basic/hss/tmp</filetree>
  <filetree root="/opt/OpenIMSCore/FHoSS/deploy" seq="start">./files/ims-basic/hss/opt.OpenIMSCore</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/mysql restart &amp;&amp; /tmp/hss_db_create.sh</exec>
  <!--exec seq="start" type="verbatim">cd /opt/OpenIMSCore/FHoSS/deploy &amp;&amp; ./startup.sh</exec>-->
  <exec seq="start" type="verbatim">/etc/init.d/FHoSS_ini start</exec>
  <exec seq="initial" type="verbatim">/tmp/hss_db_create.sh</exec>
  <exec seq="users" type="verbatim">/tmp/hss_import.sh /tmp/users.sql</exec>
  <exec seq="presence" type="verbatim">/tmp/hss_import.sh /tmp/presence.sql</exec>
  <exec seq="voicemail_ifc" type="verbatim">/tmp/hss_import.sh /tmp/voicemail_ifc.sql</exec>
  <exec seq="voicemail_route" type="verbatim">/tmp/hss_import.sh /tmp/voicemail_route.sql</exec>
  <exec seq="complete" type="verbatim">/tmp/hss_import.sh /tmp/complete.sql</exec>
</vm>

<vm name="voice">
  <mem>256M</mem>
  <console id="0">pts</console>
  <console id="1">pts</console>
  <if id="1" net="Net0">
      <ipv4>203.0.113.6/24</ipv4>
  </if>
  <route type="ipv4" gw="203.0.113.1">default</route>
  <filetree root="/etc/asterisk/" seq="start">./files/ims-basic/voice/etc.asterisk/</filetree>
  <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
  <exec seq="start" type="verbatim">/etc/init.d/asterisk restart</exec>
</vm>
```

```
  <vm name="alice">
    <console id="0">pts</console>
    <console id="1">pts</console>
    <if id="1" net="Net0">
        <ipv4>203.0.113.7/24</ipv4>
    </if>
    <route type="ipv4" gw="203.0.113.1">default</route>
    <filetree root="/root" seq="start">./files/ims-basic/pjsua/</filetree>
    <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
    <filetree root="/tmp/" seq="start">./files/ims-basic/tmp/</filetree>
    <exec seq="start" type="verbatim">/etc/init.d/ims stop</exec>
    <exec seq="initial" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="users" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="presence" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="voicemail_ifc" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="voicemail_route" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="complete" type="verbatim">/tmp/kill_pjsua.sh</exec>

  </vm>
  <vm name="bob">
    <console id="0">pts</console>
    <console id="1">pts</console>
    <if id="1" net="Net0">
        <ipv4>203.0.113.8/24</ipv4>
    </if>
    <route type="ipv4" gw="203.0.113.1">default</route>
    <filetree root="/root" seq="start">./files/ims-basic/pjsua/</filetree>
    <filetree root="/etc/" seq="start">./files/ims-basic/resolvconf/</filetree>
    <filetree root="/tmp/" seq="start">./files/ims-basic/tmp/</filetree>
    <exec seq="start" type="verbatim">/etc/init.d/ims stop</exec>
    <exec seq="initial" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="users" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="presence" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="voicemail_ifc" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="voicemail_route" type="verbatim">/tmp/kill_pjsua.sh</exec>
    <exec seq="complete" type="verbatim">/tmp/kill_pjsua.sh</exec>
  </vm>

</vnuml>
```

C.1.2. Fichero de resolución DNS: /etc/resolv.conf

```
search example.org
options timeout:1 attempts:2
nameserver 203.0.113.1
```

C.1.3. Script de creación de la base de datos general para kamailio: /tmp/kam_db_create.sh

```
#!/usr/bin/expect -f
spawn kamdbctl create
expect "MySQL password for root: "
send "\r"
expect "Install presence related tables? (y/n): "
send "y\r"
expect "rtpproxy? (y/n): "
send "y\r"
expect "uid_uri_db? (y/n): "
send "y\r"
expect "# "
```

C.2. I-CSCF

C.2.1. Fichero de definición y contenido de la base de datos del I-CSCF: /etc/kamailio/icscf.mysql.sql

```
-- MySQL dump 10.9
--
-- Host: localhost    Database: icscf
-- ------------------------------------------------------
-- Server version       4.1.20-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
```

```
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Current Database: `icscf`
--

/*!40000 DROP DATABASE IF EXISTS `icscf`*/;

CREATE DATABASE /*!32312 IF NOT EXISTS*/ `icscf` /*!40100 DEFAULT CHARACTER SET utf8 */;

USE `icscf`;

--
-- Table structure for table `nds_trusted_domains`
--

DROP TABLE IF EXISTS `nds_trusted_domains`;
CREATE TABLE `nds_trusted_domains` (
  `id` int(11) NOT NULL auto_increment,
  `trusted_domain` varchar(83) NOT NULL default '',
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Table structure for table `s_cscf`
--

DROP TABLE IF EXISTS `s_cscf`;
CREATE TABLE `s_cscf` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(83) NOT NULL default '',
  `s_cscf_uri` varchar(83) NOT NULL default '',
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Table structure for table `s_cscf_capabilities`
--

DROP TABLE IF EXISTS `s_cscf_capabilities`;
CREATE TABLE `s_cscf_capabilities` (
  `id` int(11) NOT NULL auto_increment,
  `id_s_cscf` int(11) NOT NULL default '0',
  `capability` int(11) NOT NULL default '0',
  PRIMARY KEY  (`id`),
  KEY `idx_capability` (`capability`),
  KEY `idx_id_s_cscf` (`id_s_cscf`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- MySQL dump 10.9
--
-- Host: localhost    Database: icscf
-- ------------------------------------------------------
-- Server version       4.1.20-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Current Database: `icscf`
--

CREATE DATABASE /*!32312 IF NOT EXISTS*/ `icscf` /*!40100 DEFAULT CHARACTER SET utf8 */;

USE `icscf`;

--
-- Dumping data for table `nds_trusted_domains`
--
```

```
/*!40000 ALTER TABLE `nds_trusted_domains` DISABLE KEYS */;
LOCK TABLES `nds_trusted_domains` WRITE;
INSERT INTO `nds_trusted_domains` VALUES (1,éxample.org');
UNLOCK TABLES;
/*!40000 ALTER TABLE `nds_trusted_domains` ENABLE KEYS */;

--
-- Dumping data for table `s_cscf`
--


/*!40000 ALTER TABLE `s_cscf` DISABLE KEYS */;
LOCK TABLES `s_cscf` WRITE;
INSERT INTO `s_cscf` VALUES (1,'First S-CSCF','sip:scscf.example.org:5060'), (2,'Second S-CSCF','sip:scscf2.example.org:5060'
UNLOCK TABLES;
/*!40000 ALTER TABLE `s_cscf` ENABLE KEYS */;

--
-- Dumping data for table `s_cscf_capabilities`
--


/*!40000 ALTER TABLE `s_cscf_capabilities` DISABLE KEYS */;
LOCK TABLES `s_cscf_capabilities` WRITE;
INSERT INTO `s_cscf_capabilities` VALUES (1,1,0),(2,1,1),(3,2,0),(4,2,1);
UNLOCK TABLES;
/*!40000 ALTER TABLE `s_cscf_capabilities` ENABLE KEYS */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

# DB access rights
grant delete,insert,select,update on icscf.* to kamailio@localhost identified by 'kamailiorw';
/* grant delete,insert,select,update on icscf.* to provisioning@localhost identified by 'provi'; */
```

C.2.2. Script para la generación de la base de datos del I-CSCF: /tmp/kam_icscf_create.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost -Bse "source /etc/kamailio/icscf.mysql.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.2.3. Fichero de configuración local del I-CSCF: /etc/kamailio/icscf.cfg

```
# SIP / UDP
listen=udp:203.0.113.4:5060
# SIP / TCP
listen=tcp:203.0.113.4:5060
...

alias=ims.example.org
alias=icscf.example.org

#!define NETWORKNAME "ims.example.org"
#!define HOSTNAME "icscf.example.org"

# Connection URL for the database:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/icscf"

# Allowed IPs for XML-RPC-Queries
#!define XMLRPC\_WHITELIST\_1 "127.0.0.1"

# Enabled Features for this host:
#!define WITH_XMLRPC
```

C.2.4. Configuración del dominio SIP de servicio y del motor de la base de datos: /etc/kamailio/kamctlrc

```
## your SIP domain
# OURCHANGES Start
```

```
#configured example.org
SIP_DOMAIN=example.org
# OURCHANGES End

# If you want to setup a database with kamdbctl, you must at least specify
# this parameter.
# OURCHANGES Start
# uncommented for using mysql
DBENGINE=MYSQL
# OURCHANGES End
```

C.2.5. Fichero configuración de la interfaz Diameter del I-CSCF: /etc/kamailio/icscf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DiameterPeer
        FQDN="icscf.example.org"
        Realm="example.org"
        Vendor_Id="10415"
        Product_Name="CDiameterPeer"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="8"
        TransactionTimeout="5"
        SessionsHashSize="128"
        DefaultAuthSessionTimeout="3600"
        MaxAuthSessionTimeout="3600"
>

        <Peer FQDN="hss.example.org" Realm="example.org" port="3868"/>

        <Acceptor port="3868" bind="203.0.113.4"/>

        <Auth id="16777216" vendor="10415"/> <!--3GPP CxDX -->

        <DefaultRoute FQDN="hss.example.org" metric="10"/>
</DiameterPeer>
```

C.2.6. Fichero de definición de las opciones de inicio de kamailio: /etc/default/kamailio

```
#
# Kamailio startup options
#

# OURCHANGES Start
# Set to yes to enable kamailio, once configured properly.
RUN_KAMAILIO=yes
# OURCHANGES End

# User to run as
#USER=kamailio

# Group to run as
#GROUP=kamailio

# Amount of shared and private memory to allocate
# for the running Kamailio server (in Mb)
#SHM_MEMORY=64
#PKG_MEMORY=8

# Config file
#CFGFILE=/etc/kamailio/kamailio.cfg

# Enable the server to leave a core file when it crashes.
# Set this to 'yes' to enable Kamailio to leave a core file when it crashes
# or 'no' to disable this feature. This option is case sensitive and only
# accepts 'yes' and 'no' and only in lowercase letters.
# On some systems it is necessary to specify a directory for the core files
# to get a dump. Look into the kamailio init file for an example configuration.
#DUMP_CORE=yes
```

C.3. P-CSCF

C.3.1. Fichero de definición de la base de datos del P-CSCF: /etc/kamailio/pcscf.sql

```
DROP TABLE `location`;
CREATE TABLE `location` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `domain` varchar(64) DEFAULT NULL,
  `aor` varchar(255) NOT NULL,
  `contact` varchar(255) DEFAULT NULL,
  `received` varchar(128) DEFAULT NULL,
  `received_port` int(10) unsigned DEFAULT NULL,
  `received_proto` int(10) unsigned DEFAULT NULL,
  `path` varchar(512) DEFAULT NULL,
  `rx_session_id` varchar(256) DEFAULT NULL,
  `reg_state` tinyint(4) DEFAULT NULL,
  `expires` datetime DEFAULT '2030-05-28 21:32:15',
  `service_routes` varchar(2048) DEFAULT NULL,
  `socket` varchar(64) DEFAULT NULL,
  `public_ids` varchar(2048) DEFAULT NULL,
  `security_type` int(11) DEFAULT NULL,
  `protocol` char(5) DEFAULT NULL,
  `mode` char(10) DEFAULT NULL,
  `ck` varchar(100) DEFAULT NULL,
  `ik` varchar(100) DEFAULT NULL,
  `ealg` char(20) DEFAULT NULL,
  `ialg` char(20) DEFAULT NULL,
  `port_uc` int(11) unsigned DEFAULT NULL,
  `port_us` int(11) unsigned DEFAULT NULL,
  `spi_pc` int(11) unsigned DEFAULT NULL,
  `spi_ps` int(11) unsigned DEFAULT NULL,
  `spi_uc` int(11) unsigned DEFAULT NULL,
  `spi_us` int(11) unsigned DEFAULT NULL,
  `t_security_type` int(11) DEFAULT NULL,
  `t_port_uc` int(11) unsigned DEFAULT NULL,
  `t_port_us` int(11) unsigned DEFAULT NULL,
  `t_spi_pc` int(11) unsigned DEFAULT NULL,
  `t_spi_ps` int(11) unsigned DEFAULT NULL,
  `t_spi_uc` int(11) unsigned DEFAULT NULL,
  `t_spi_us` int(11) unsigned DEFAULT NULL,
  `t_protocol` char(5) DEFAULT NULL,
  `t_mode` char(10) DEFAULT NULL,
  `t_ck` varchar(100) DEFAULT NULL,
  `t_ik` varchar(100) DEFAULT NULL,
  `t_ealg` char(20) DEFAULT NULL,
  `t_ialg` char(20) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `aor` (`aor`)
);
```

C.3.2. Script de generación de la base de datos de los P-CSCF: /tmp/script-kam-pcscf-create.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost kamailio -Bse "source /etc/kamailio/pcscf.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.3.3. Script de regeneración de la base de datos de los P-CSCF: /tmp/script-kam-pcscf-restore.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost kamailio -Bse "source /tmp/kam_pcscf_restore.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.3.4. Cambios en el fichero de configuración general de los P-CSCF: /etc/kamailio/kamailio.cfg

A continuación se muestran sólo los cambios aplicados en el fichero de configuración original de kamailio:

```
:16
include_file "pcscf.cfg"

:161
# Marga: Adapto el mpath a nuestro escenario
#mpath="/usr/local/lib64/kamailio/modules/"
mpath="/usr/lib64/kamailio/modules_k/:/usr/lib64/kamailio/modules/:/usr/lib/kamailio/modules_k/:\
/usr/lib/kamailio/modules/"

:505
```

```
# LAB - Modificamos el routing para que los usuarios de la plataforma puedan recibir llamadas...
# if (!ds_is_from_list()) {
#               # Originating from Subscriber:
#               route(Orig_Initial);
#       } else {
#               # Terminating to Subscriber:
#               route(Term_Initial);
#       }

        if ($route_uri =~ "sip:term@"+".*") {
                # Terminating to Subscriber:
                route(Term_Initial);
        } else {
                # Originating from Subscriber:
                route(Orig_Initial);
        }
# Fi LAB
```

C.3.5. Fichero de configuración local del P-CSCF: /etc/kamailio/pcscf.cfg

```
# IP-Adress for incoming SIP-Traffic, in the following format:

# SIP / UDP
listen=udp:203.0.113.2:5060
listen=tcp:203.0.113.2:5060
# SIP / TCP (Monitoring)
listen=tcp:127.0.0.1:5060

alias=pcscf.example.org
alias=proxy.example.org

# Port, where we listen to Traffic
#!define PORT 5060

#!subst "/NETWORKNAME/example.org/"
#!subst "/HOSTNAME/pcscf.example.org/"
#!define HOSTNAME_IP pcscf.example.org
#!define HOSTNAME_ESC "pcscf\.example\.org"

# Allowed IPs for XML-RPC-Queries
#!define XMLRPC_WHITELIST_1 "127.0.0.1"

# Databases:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/kamailio"

# IP-Adress(es) of the RTP-Proxy
#!define RTPPROXY_ADDRESS "udp:localhost:22222"

# Enabled Features for this host:
#!define WITH_NAT
#!define WITH_XMLRPC
#!define WITH_ANTIFLOOD
#!define WITH_IMS_HDR_CACHE
#!define WITH_NATPING
```

C.3.6. Fichero de configuración local del P-CSCF2: /etc/kamailio/pcscf.cfg

```
# IP-Adress for incoming SIP-Traffic, in the following format:

# SIP / UDP
listen=udp:203.0.113.12:5060
listen=tcp:203.0.113.12:5060
# SIP / TCP (Monitoring)
listen=tcp:127.0.0.1:5060

alias=pcscf2.example.org
alias=proxy2.example.org

# Port, where we listen to Traffic
#!define PORT 5060

#!subst "/NETWORKNAME/example.org/"
#!subst "/HOSTNAME/pcscf2.example.org/"
#!define HOSTNAME_IP pcscf2.example.org
#!define HOSTNAME_ESC "pcscf2\.example\.org"

# Databases:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/kamailio"
```

```
#!define WITH_NAT
#!define WITH_ANTIFLOOD
#!define WITH_IMS_HDR_CACHE
#!define WITH_NATPING
```

C.3.7. Fichero de definición de las opciones de inicio de kamailio: /etc/default/kamailio

```
# OURCHANGES Start
# Set to yes to enable kamailio, once configured properly.
RUN_KAMAILIO=yes
# OURCHANGES End
```

C.4. S-CSCF

C.4.1. Fichero de definición y contenido de la base de datos de los S-CSCF: /etc/kamailio/scscf.sql

```
INSERT INTO version (table_name, table_version) values ('contact','6');
CREATE TABLE `contact` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `contact` char(255) NOT NULL,
  `params` varchar(255) DEFAULT NULL,
  `path` varchar(255) DEFAULT NULL,
  `received` varchar(255) DEFAULT NULL,
  `user_agent` varchar(255) DEFAULT NULL,
  `expires` datetime DEFAULT NULL,
  `callid` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `contact` (`contact`)
);

INSERT INTO version (table_name, table_version) values ('impu','6');
CREATE TABLE `impu` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `impu` char(64) NOT NULL,
  `barring` int(1) DEFAULT '0',
  `reg_state` int(11) DEFAULT '0',
  `ccf1` char(64) DEFAULT NULL,
  `ccf2` char(64) DEFAULT NULL,
  `ecf1` char(64) DEFAULT NULL,
  `ecf2` char(64) DEFAULT NULL,
  `ims_subscription_data` blob,
  PRIMARY KEY (`id`),
  UNIQUE KEY `impu` (`impu`)
);
INSERT INTO version (table_name, table_version) values ('impu_contact','6');
CREATE TABLE `impu_contact` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `impu_id` int(11) NOT NULL,
  `contact_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `impu_id` (`impu_id`,`contact_id`)
);

DROP TABLE subscriber;
CREATE TABLE `subscriber` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `watcher_uri` varchar(50) NOT NULL,
  `watcher_contact` varchar(50) NOT NULL,
  `presentity_uri` varchar(50) NOT NULL,
  `event` int(11) NOT NULL,
  `expires` datetime NOT NULL,
  `version` int(11) NOT NULL,
  `local_cseq` int(11) NOT NULL,
  `call_id` varchar(50) NOT NULL,
  `from_tag` varchar(50) NOT NULL,
  `to_tag` varchar(50) NOT NULL,
  `record_route` varchar(50) NOT NULL,
  `sockinfo_str` varchar(50) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `watcher_uri` (`event`,`watcher_contact`,`presentity_uri`)
);

INSERT INTO version (table_name, table_version) values ('impu_subscriber','6');
CREATE TABLE `impu_subscriber` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `impu_id` int(11) NOT NULL,
  `subscriber_id` int(11) NOT NULL,
```

```
  PRIMARY KEY (`id`),
  UNIQUE KEY `impu_id` (`impu_id`,`subscriber_id`)
);
```

C.4.2. Script para la generación de la base de datos de los S-CSCF: /tmp/kam_scscf_create.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost kamailio -Bse "source /etc/kamailio/scscf.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.4.3. Script para la regeneración de la base de datos de los S-CSCF: /tmp/kam_scscf_restore.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost kamailio -Bse "source /tmp/kam_scscf_restore.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.4.4. Cambios en el fichero de configuración general de los S-CSCF: /etc/kamailio/kamailio.cfg

A continuación se muestran sólo los cambios en el fichero de configuración original:

```
:36
include_file "scscf.cfg"

:229
# -- CDP params --
modparam("cdp","config_file","/etc/kamailio/scscf.xml")

:931
        #LAB - Inicio aplicar ruta error Voicemail para INVITE
        if (is_method("INVITE")) {
                #Introducimos la ruta al Voicemail en caso de error
                if(!t_is_set("failure_route")) t_on_failure("VOICEMAIL");
                if (impu_registered("location")) {
                        #Modificamos el timeout de llamada a 8 segundos
                        #para usuarios registrados en la plataforma
                       t_set_fr(8000);
                }
        }
        #Fi LAB - Fin aplicar ruta error Voicemail para INVITE

:1048
#LAB - Ruta al Voicemail
failure_route[VOICEMAIL] {
        xlog("L_ERR","User unavailable, voicemail route forced 2\n\n");
        if (impu_registered("location")) {
                rewritehostport("voicemail.example.org:5060");
                t_relay();
        }
}
#Fi LAB
```

C.4.5. Fichero de configuración local del S-CSCF: /etc/kamailio/scscf.cfg

```
# IP-Adress for incoming SIP-Traffic, in the following format:
listen=udp:203.0.113.3:5060
listen=tcp:203.0.113.3:5060

#!define NETWORKNAME "example.org"
#!define NETWORKNAME_ESC "example\.org"
#!define HOSTNAME "scscf.example.org"
#!define URI "sip:scscf.example.org:5060"
#!define HOSTNAME_ESC "scscf\.example\.org"

alias=scscf.example.org

# ENUM-Server to query:
#!define ENUM_SUFFIX "example.org."
```

```
# Connection URL for the database:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/kamailio"

# Select Authorization Algorhithm:
# Let the HSS decide
#!define REG_AUTH_DEFAULT_ALG "HSS-Selected"

# Number of TCP Processes
#!define TCP_PROCESSES 3

#!define XMLRPC_WHITELIST_1 "127.0.0.1"
#!define XMLRPC_WHITELIST_2 "203.0.113.3"

# Enabled Features for this host:
#!define WITH_XMLRPC
```

C.4.6. Fichero de configuración local del S-CSCF2: /etc/kamailio/scscf.cfg

```
# IP-Adress for incoming SIP-Traffic, in the following format:
listen=udp:203.0.113.9:5060
listen=tcp:203.0.113.9:5060

#!define NETWORKNAME "example.org"
#!define NETWORKNAME_ESC "example\.org"
#!define HOSTNAME "scscf2.example.org"
#!define URI "sip:scscf2.example.org:5060"
#!define HOSTNAME_ESC "scscf2\.example\.org"

alias=scscf2.example.org

# ENUM-Server to query:
#!define ENUM_SUFFIX "example.org."

# Connection URL for the database:
#!define DB_URL "mysql://kamailio:kamailiorw@localhost/kamailio"

# Select Authorization Algorhithm:
# Let the HSS decide
#!define REG_AUTH_DEFAULT_ALG "HSS-Selected"

# Number of TCP Processes
#!define TCP_PROCESSES 3

#!define XMLRPC_WHITELIST_1 "127.0.0.1"
#!define XMLRPC_WHITELIST_2 "203.0.113.9"

# Enabled Features for this host:
#!define WITH_XMLRPC
```

C.4.7. Fichero configuración de la interfaz Diameter del S-CSCF: /etc/kamailio/scscf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DiameterPeer
        FQDN="scscf.example.org"
        Realm="example.org"
        Vendor_Id="10415"
        Product_Name="CDiameterPeer"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="8"
        TransactionTimeout="5"
        SessionsHashSize="128"
        DefaultAuthSessionTimeout="3600"
        MaxAuthSessionTimeout="3600"
>
        <Peer FQDN="hss.example.org" Realm="example.org" port="3868"/>
        <Acceptor port="3868" bind="203.0.113.3"/>
        <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
        <Auth id="4" vendor="10415"/> <!--3GPP Ro -->
        <DefaultRoute FQDN="hss.example.org" metric="10"/>
</DiameterPeer>
```

C.4.8. Fichero configuración de la interfaz Diameter del S-CSCF2: /etc/kamailio/scscf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DiameterPeer
        FQDN="scscf2.example.org"
        Realm="example.org"
        Vendor_Id="10415"
        Product_Name="CDiameterPeer"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="8"
        TransactionTimeout="5"
        SessionsHashSize="128"
        DefaultAuthSessionTimeout="3600"
        MaxAuthSessionTimeout="3600"
>
        <Peer FQDN="hss.example.org" Realm="example.org" port="3868"/>
        <Acceptor port="3868" bind="203.0.113.9"/>
        <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
        <Auth id="4" vendor="10415"/> <!--3GPP Ro -->
        <DefaultRoute FQDN="hss.example.org" metric="10"/>
</DiameterPeer>
```

C.4.9. Fichero de definición de las opciones de inicio de kamailio: /etc/default/kamailio

```
#
# Kamailio startup options
#

# OURCHANGES Start
# Set to yes to enable kamailio, once configured properly.
RUN_KAMAILIO=yes
# OURCHANGES End

# User to run as
#USER=kamailio

# Group to run as
#GROUP=kamailio

# Amount of shared and private memory to allocate
# for the running Kamailio server (in Mb)
#SHM_MEMORY=64
#PKG_MEMORY=8

# Config file
#CFGFILE=/etc/kamailio/kamailio.cfg

# Enable the server to leave a core file when it crashes.
# Set this to 'yes' to enable Kamailio to leave a core file when it crashes
# or 'no' to disable this feature. This option is case sensitive and only
# accepts 'yes' and 'no' and only in lowercase letters.
# On some systems it is necessary to specify a directory for the core files
# to get a dump. Look into the kamailio init file for an example configuration.
#DUMP_CORE=yes
```

C.4.10. Configuración del dominio SIP de servicio y del motor de la base de datos: /etc/kamailio/kamctlrc

```
## your SIP domain
# OURCHANGES Start
#configured example.org
SIP_DOMAIN=example.org
# OURCHANGES End

# If you want to setup a database with kamdbctl, you must at least specify
# this parameter.
# OURCHANGES Start
# uncommented for using mysql
DBENGINE=MYSQL
# OURCHANGES End
```

C.5. HSS

C.5.1. Script de generación de la base de datos del HSS: /tmp/hss_db_create.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost -Bse "source /tmp/hss_db.sql"
expect "Enter password:"
send "\r"
expect "# "

spawn mysql -u root -p -h localhost hss_db -Bse "source /tmp/asterisk.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.5.2. Script de regeneración de la base de datos del HSS: /tmp/hss_import.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost -Bse "source /tmp/hss_db.sql"
expect "Enter password:"
send "\r"
expect "# "

spawn mysql -u root -p -h localhost hss_db -Bse "source /tmp/asterisk.sql"
expect "Enter password:"
send "\r"
expect "# "

if {$argc != 0} {
        set data [lindex $argv 0]
        spawn mysql -u root -p -h localhost hss_db -Bse "source $data"
        expect "Enter password:"
        send "\r"
        expect "# "
}
```

C.5.3. Punto de regeneración de la base de datos presence_ini: /tmp/presence_ini.sql

A continuación se muestran sólo los cambios respecto a la definición inicial de la base de datos del HSS:

```
INSERT INTO `impi` VALUES (5,3,'alice@example.org','alice',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0',\\
'000000000000','','',0,3600,1),(6,4,'bob@example.org','bob',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1),\\
(7,5,'claire@example.org','claire',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0',\\
'000000000000','','',0,3600,1);
INSERT INTO `impi_impu` VALUES (5,5,3,0),(6,6,4,0),(7,7,5,0);
INSERT INTO `impu` VALUES (3,'sip:alice@example.org',0,0,0,2,3,-1,'','',0,1),\\
(4,'sip:bob@example.org',0,0,0,2,4,-1,'','',0,1),\\
(5,'sip:claire@example.org',0,0,0,2,5,-1,'','',0,1);
INSERT INTO `impu_visited_network` VALUES (3,3,2),(4,4,2),(5,5,2);
INSERT INTO `imsu` VALUES (3,'Alice','','',-1,1),(4,'Bob','','',-1,2),(5,'Claire','','',-1,2);
INSERT INTO `preferred_scscf_set` VALUES (8,2,'scscf2.example.org','sip:scscf2.example.org:5060',0),\\
(7,1,'scscf.example.org','sip:scscf2.example.org:5060',1),\\
(6,1,'scscf.example.org','sip:scscf.example.org:5060',0),\\
(9,2,'scscf2.example.org','sip:scscf.example.org:5060',1);
INSERT INTO `sp` VALUES (2,'sp1',0);
INSERT INTO `visited_network` VALUES (2,'example.org');
```

C.5.4. Punto de regeneración de la base de datos voicemail_ini: /tmp/voicemail_ini.sql

A continuación se muestran sólo los cambios respecto a la definición inicial de la base de datos del HSS:

```
INSERT INTO `application_server` VALUES (3,'Presence_as','sip:presence.example.org:5060',0,'',\
'presence.example.org',1024,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
INSERT INTO `ifc` VALUES (2,'Presence_ifc',3,2,-1);
INSERT INTO `impi` VALUES (5,3,'alice@example.org','alice',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0',\\
'000000000000','','',0,3600,1),(6,4,'bob@example.org','bob',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1),\\
(7,5,'claire@example.org','claire',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0',\\
'000000000000','','',0,3600,1);
INSERT INTO `impi_impu` VALUES (5,5,3,0),(6,6,4,0),(7,7,5,0);
INSERT INTO `impu` VALUES (3,'sip:alice@example.org',0,0,0,3,3,-1,'','',0,1),\\
(4,'sip:bob@example.org',0,0,0,3,4,-1,'','',0,1),\\
(5,'sip:claire@example.org',0,0,0,3,5,-1,'','',0,1);
INSERT INTO `impu_visited_network` VALUES (3,3,2),(4,4,2),(5,5,2);
INSERT INTO `imsu` VALUES (3,'Alice','','',-1,1),(4,'Bob','','',-1,2),(5,'Claire','','',-1,2);
INSERT INTO `preferred_scscf_set` VALUES (8,2,'scscf2.example.org','sip:scscf2.example.org:5060',0),\\
(7,1,'scscf.example.org','sip:scscf2.example.org:5060',1),\\
```

```
(6,1,'scscf.example.org','sip:scscf.example.org:5060',0),\\
(9,2,'scscf2.example.org','sip:scscf.example.org:5060',1);
INSERT INTO `sp` VALUES (2,'sp1',0),(3,'ims_services',0);
INSERT INTO `sp_ifc` VALUES (2,3,2,0);
INSERT INTO `spt` VALUES (16,2,0,0,1,NULL,'PUBLISH',NULL,NULL,NULL,NULL,NULL,0),\\
(17,2,0,1,1,NULL,'SUBSCRIBE',NULL,NULL,NULL,NULL,NULL,0);
INSERT INTO `tp` VALUES (2,'Presence_tp',0);
INSERT INTO `visited_network` VALUES (2,'example.org');
```

C.5.5. Punto de regeneración de la base de datos voicemail_route: /tmp/voicemail_route.sql

A continuación se muestran sólo los cambios respecto a la definición inicial de la base de datos del HSS:

```
INSERT INTO `application_server` VALUES (3,'Presence_as','sip:presence.example.org:5060',0,'',\\
'presence.example.org',1024,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\\
0,0,0,0,0,0,0,0,0,0,0,0),(4,'Voicemail_as','sip:voicemail.example.org:5060',\\
0,'','voicemail.example.org',1024,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0);
INSERT INTO `ifc` VALUES (2,'Presence_ifc',3,2,-1),(3,'Voicemail_ifc',4,3,-1);
INSERT INTO `impi` VALUES (5,3,'alice@example.org','alice',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1),\\
(6,4,'bob@example.org','bob',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','\\
'000000000000','','',0,3600,1),\\
(7,5,'claire@example.org','claire',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1);
INSERT INTO `impi_impu` VALUES (5,5,3,0),(6,6,4,0),(7,7,5,0);
INSERT INTO `impu` VALUES (3,'sip:alice@example.org',0,0,0,3,3,-1,'','',0,1),\\
(4,'sip:bob@example.org',0,0,0,3,4,-1,'','',0,1),\\
(5,'sip:claire@example.org',0,0,0,3,5,-1,'','',0,1);
INSERT INTO `impu_visited_network` VALUES (3,3,2),(4,4,2),(5,5,2);
INSERT INTO `imsu` VALUES (3,'Alice','','',-1,1),(4,'Bob','','',-1,2),(5,'Claire','','',-1,2);
INSERT INTO `preferred_scscf_set` VALUES (8,2,'scscf2.example.org','sip:scscf2.example.org:5060',0),\\
(7,1,'scscf.example.org','sip:scscf2.example.org:5060',1),\\
(6,1,'scscf.example.org','sip:scscf.example.org:5060',0),\\
(9,2,'scscf2.example.org','sip:scscf.example.org:5060',1);
INSERT INTO `sp` VALUES (2,'sp1',0),(3,'ims_services',0);
INSERT INTO `sp_ifc` VALUES (2,3,2,0),(3,3,3,1);
INSERT INTO `spt` VALUES (16,2,0,0,1,NULL,'PUBLISH',NULL,NULL,NULL,NULL,NULL,0),\\
(17,2,0,1,1,NULL,'SUBSCRIBE',NULL,NULL,NULL,NULL,NULL,0),\\
(19,3,0,0,3,NULL,NULL,NULL,NULL,2,NULL,NULL,0),\\
(20,3,0,0,1,NULL,'INVITE',NULL,NULL,NULL,NULL,NULL,-2);
INSERT INTO `tp` VALUES (2,'Presence_tp',0),(3,'Voicemail_tp',0);
INSERT INTO `visited_network` VALUES (2,'example.org');
```

C.5.6. Punto de regeneración de la base de datos complete: /tmp/complete.sql

A continuación se muestran sólo los cambios respecto a la definición inicial de la base de datos del HSS:

```
INSERT INTO `application_server` VALUES (3,'Presence_as','sip:presence.example.org:5060',0,'',\
'presence.example.org',1024,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),\\
(4,'Voicemail_as','sip:voicemail.example.org:5060',0,'','voicemail.example.org'\\
,1024,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
INSERT INTO `ifc` VALUES (2,'Presence_ifc',3,2,-1),(3,'Voicemail_ifc',4,3,-1);
INSERT INTO `impi` VALUES (5,3,'alice@example.org','alice',255,4,'\0\0','\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0',\\
'000000000000','','',0,3600,1),(6,4,'bob@example.org','bob',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1),\\
(7,5,'claire@example.org','claire',255,4,'\0\0',\\
'\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0','000000000000','','',0,3600,1);
INSERT INTO `impi_impu` VALUES (5,5,3,0),(6,6,4,0),(7,7,5,0);
INSERT INTO `impu` VALUES (3,'sip:alice@example.org',0,0,0,3,3,-1,'','',0,1),(4,'sip:bob@example.org',\\
0,0,0,3,4,-1,'','',0,1),(5,'sip:claire@example.org',0,0,0,3,5,-1,'','',0,1);
INSERT INTO `impu_visited_network` VALUES (3,3,2),(4,4,2),(5,5,2);
INSERT INTO `imsu` VALUES (3,'Alice','','',-1,1),(4,'Bob','','',-1,2),(5,'Claire','','',-1,2);
INSERT INTO `preferred_scscf_set` VALUES (8,2,'scscf2.example.org','sip:scscf2.example.org:5060',0),\\
(7,1,'scscf.example.org','sip:scscf2.example.org:5060',1),\\
(6,1,'scscf.example.org','sip:scscf.example.org:5060',0),\\
(9,2,'scscf2.example.org','sip:scscf.example.org:5060',1),\\
(10,3,'voicemail','sip:voicemail.example.org:5060',1);
INSERT INTO `sp` VALUES (2,'sp1',0),(3,'ims_services',0);
INSERT INTO `sp_ifc` VALUES (2,3,2,0),(3,3,3,1);
INSERT INTO `spt` VALUES (16,2,0,0,1,NULL,'PUBLISH',NULL,NULL,NULL,NULL,NULL,0),\\
(17,2,0,1,1,NULL,'SUBSCRIBE',NULL,NULL,NULL,NULL,NULL,0),\\
(19,3,0,0,3,NULL,NULL,NULL,NULL,2,NULL,NULL,0),\\
(20,3,0,0,1,NULL,'INVITE',NULL,NULL,NULL,NULL,NULL,0),\\
(21,3,0,1,1,NULL,'INVITE',NULL,NULL,NULL,NULL,NULL,0),\\
(22,3,0,1,0,'sip:*86@example.org',NULL,NULL,NULL,NULL,NULL,NULL,0);
```

```
INSERT INTO `tp` VALUES (2,'Presence_tp',0),(3,'Voicemail_tp',0);
INSERT INTO `visited_network` VALUES (2,'example.org')
```

C.5.7. Fichero configuración de la interfaz Diameter del HSS: /opt/OpenIMSCore/FHoSS/deploy/DiameterPeerHSS.xml

Listing C.1: conf-DiameterPeerHSS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- HSS Server config -->
<DiameterPeer
        FQDN="hss.example.org"
        Realm="example.org"
        Vendor_Id="10415"
        Product_Name="JavaDiameterPeer"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="32"
>
        <Peer FQDN="icscf.example.org" Realm="example.org" port="3868" />
        <Peer FQDN="scscf.example.org" Realm="example.org" port="3868" />
        <Peer FQDN="scscf2.example.org" Realm="example.org" port="3868" />
        <Acceptor port="3868" bind="203.0.113.5" />
        <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
        <Auth id="16777216" vendor="4491"/><!-- CableLabs Cx -->
        <Auth id="16777216" vendor="13019"/><!-- ETSI/TISPAN Cx -->
        <Auth id="16777216" vendor="0"/><!-- ETSI/TISPAN Cx -->
        <Auth id="16777217" vendor="10415"/><!-- 3GPP Sh -->
        <Auth id="16777221" vendor="10415"/>
</DiameterPeer>
```

C.5.8. Fichero configuración de la interfaz Diameter del HSS: /opt/OpenIMSCore/FHoSS/deploy/hibernate.properties

```
## MySQL

# hibernate configuration
hibernate.dialect=org.hibernate.dialect.MySQLDialect
#hibernate.connection.driver_class=org.gjt.mm.mysql.Driver
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://127.0.0.1:3306/hss_db
hibernate.connection.username=hss
hibernate.connection.password=hss
hibernate.connection.isolation=1

# C3P0 configuration
hibernate.c3p0.acquire_increment=1
hibernate.c3p0.min_size=1
hibernate.c3p0.max_size=30
hibernate.c3p0.timeout=3600
hibernate.c3p0.max_statements=0
hibernate.c3p0.idle_test_period=1200
```

C.5.9. Fichero configuración de la interfaz Diameter del HSS: /opt/OpenIMSCore/FHoSS/deploy/hss.properties

```
# FOKUS HSS Properties file
#-----------------------------------------------------------------------------------------------------------------
# host & port : specify the IP address and the port where Tomcat is listening, e.g. host=127.0.0.1; port=8080;

host=203.0.113.5
port=8080

# Authentication properties
#-----------------------------------------------------------------------------------------------------------------

# default operator and amf values
#-----------------------------------------------------------------------------------------------------------------
# operator_id, as hex bytes, required length 32 byte,
# e.g. 00000000000000000000000000000000
operatorId=00000000000000000000000000000000
# amf_id: Default amf id as hex bytes, required length 4 byte, e.g. 0000
amfId=0000

# configuration parameters relating to Milenage algorithm
```

```
#-------------------------------------------------------------------------------------------------------------------

# Enable or disable the use of AK in the Milenage algorithm; if this flag is enabled,
#then is mandatory to be enabled also on the client side
USE_AK=true

# IND_LEN property - contains the number of bits assigned for the Index; it is used in the generation \
process of new SQN values
# We are using SQN values which are not time based, as is specified here C.1.1.2, C.1.2, C.2, C3.2 and \
C.3.4 of TS 33.102
# (SQN = SEQ || IND)
IND_LEN=5

# delta value, assuring the protection against wrap around counter in the USIM
delta=268435456

# L - limit on the difference between SEQ_MS (Mobile Station) and SEQ_HE (HSS)
L=32

# Sh-Settings
#-------------------------------------------------------------------------------------------------------------------

# Enable or disable IFC Notification mechanism. If you need this feature please enable it. However, be aware \
that this feature imply
#important time for processing as more validation is required every time after an update (for entities as: \
IFC, TP, SPT, AS, SP_IFC),
# and could affect the web console interface response-ness.
iFC_NOTIF_ENABLED=false
# interval to check in the db if there are any notifications to push over Sh
#SH_NOTIF_CHECK_INTERVAL=10
SH_NOTIF_CHECK_INTERVAL=1

# Cx-Settings
# whether to automatically enable a PPR on each IMPU update. Probably not a good idea.
AUTO_PPR_ENABLED=false
# interval to check in the db if there are any events to push over Cx
CX_EVENT_CHECK_INTERVAL=1

# Expiry Time limit - indicates the subscriptions maximum lifetime allowed by the HSS
expiry_time_lim=3600
```

C.5.10. Fichero configuración de la interfaz Diameter del HSS: /opt/OpenIMSCore/FHoSS/deploy/log4j.properties

```
# Set root category priority to INFO and its only appender to CONSOLE.
#log4j.rootCategory=ERROR, MYCONSOLE
log4j.rootLogger=WARN, MYCONSOLE, LOGFILE, LOGFILE2
log4j.logger.de.fhg.fokus.diameter=INFO
log4j.logger.de.fhg.fokus.hss=DEBUG

# MYCONSOLE is set to be a ConsoleAppender using a PatternLayout.
log4j.appender.MYCONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.MYCONSOLE.Threshold=DEBUG
log4j.appender.MYCONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.MYCONSOLE.layout.ConversionPattern=%d %-5p %c - %M %m%n

# LOGFILE is set to be a File appender using a PatternLayout.
log4j.appender.LOGFILE=org.apache.log4j.FileAppender
log4j.appender.LOGFILE.File=logs/hss.server.log
log4j.appender.LOGFILE.Append=true
log4j.appender.LOGFILE.Threshold=WARN
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%d [%t] %-5p %c %x - %m%n

# LOGFILE2
log4j.appender.LOGFILE2=org.apache.log4j.DailyRollingFileAppender
# Required! Specify here the path to the log file
log4j.appender.LOGFILE2.File=logs/hss.activities.log
log4j.appender.LOGFILE2.Append=true
log4j.appender.LOGFILE2.Threshold=DEBUG
log4j.appender.LOGFILE2.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.LOGFILE2.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE2.layout.ConversionPattern=%d [%t] %-5p %c %x - %m%n
```

C.5.11. Script de inicio: /etc/init.d/FHoSS_ini

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          openimscore-fhoss
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
```

```
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Manage OSIMS FHoSS function
### END INIT INFO

#
# Author: Arnaud Morin <arnaud1.morin@orange-ftgroup.com>
# Actual version 1.0.1
# Version-history:
#       1.0.1           2008-11-21      Arnaud Morin
#       1.0.0           2007-12-15      David Blaisonneau

# Do NOT "set -e"

# DAEMON VALUES
DESC="OpenSource IMS Core - HSS"
NAME=hss

# PATH should only include /usr/* if it runs after the mountnfs.sh script
JAVA_BIN=/usr/bin/java
PATH=/usr/sbin:/usr/bin:/sbin:/bin
EXECPATH=/opt/OpenIMSCore/FHoSS/deploy
LIBPATH=$EXECPATH/lib
LIBLOG4J=log4j.properties
PIDFILE=/var/run/$NAME.pid
#SCRIPTNAME=/etc/init.d/osims_$NAME
SCRIPTNAME=$EXECPATH/fhoss.sh
HSSCLASS=de.fhg.fokus.hss.main.HSSContainer
VERBOSE=yes

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
[ -f /etc/default/rcS ] && . /etc/default/rcS

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure that this file is present.
. /lib/lsb/init-functions

VERBOSE=YES

getCurrentPID()
{
        PID=$(ps -ef |  grep $JAVA_BIN | grep $HSSCLASS | cut -c10-15)
        echo $PID
}

#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started

        PID=$(getCurrentPID)
        [ "$PID" != "" ] && echo -n "...Already running (pid: $PID)..." && return 1

        # --make-pidfile in order to create pid automaticaly
        # --background in order to run in background
        start-stop-daemon --start --quiet --background --make-pidfile --pidfile $PIDFILE --exec $SCRIPTNAME

        #Wait for deamon
        sleep 3

        #Verification
        PID=$(getCurrentPID)
        [ "$PID" = "" ]  && return 2

        echo $PID > $PIDFILE
        return 0
}

#
# Function that stops the daemon/service
#
do_stop()
{
        # Return
        #   0 if daemon has been stopped
        #   1 if daemon was already stopped
        #   2 if daemon could not be stopped
        #   other if a failure occurred
```

```
        PID=$(getCurrentPID)
        [ "$PID" = "" ] && echo -n "Not running" && return 1

        kill -9 $PID 2>&1 > /dev/null
        sleep 3

        PID=$(getCurrentPID)
        [ "$PID" = "" ] || return 2

        rm -f $PIDFILE
        return 0
}

#
# Function that return status of the daemon/service
#
do_status()
{
        # Return
        #   0 if daemon is NOT running
        #   1 if daemon is running
        PID=$(getCurrentPID)
        [ "$PID" = "" ] && return 0
        return 1
}

if [ "$USER" != root ]
then
        echo "Please, run $SCRIPTNAME as root using 'sudo $SCRIPTNAME' command" >&2
fi
case "$1" in
  start)
        [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
        do_start
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  stop)
        [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
        do_stop
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  status)
        log_daemon_msg "Process running ??? $DESC" "$NAME"
        do_status
        case "$?" in
                0) log_end_msg 1 ;;
                *) log_end_msg 0 ;;
        esac
        ;;
  restart)
        [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
        do_stop
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
        do_start
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|status|restart}" >&2
        exit 3
        ;;
esac
```

C.5.12. Script de inicio intermedio: /opt/OpenIMSCore/FHoSS/deploy/fhoss.sh

```
#!/bin/bash
cd /opt/OpenIMSCore/FHoSS/deploy/
./startup.sh
```

C.5.13. Script de inicio original modificado: /opt/OpenIMSCore/FHoSS/deploy/startup.sh

```
#!/bin/bash
# ------------------------------------------------------------
# Include JAR Files
# ------------------------------------------------------------

echo "Building Classpath"
CLASSPATH=$CLASSPATH:log4j.properties:.
for i in lib/*.jar; do CLASSPATH="$i":"$CLASSPATH"; done
echo "Classpath is $CLASSPATH."

# ------------------------------------------------------------
# Start-up
# ------------------------------------------------------------

/usr/bin/java -cp $CLASSPATH de.fhg.fokus.hss.main.HSSContainer $1 $2 $3 $4 $5 $6 $7 $8 $9
```

C.5.14. Definició de los usuarios de la consolda de administración web: /opt/OpenIMSCore/FHoSS/tomcat/conf/tomcat-users.xml

```
<tomcat-users>
        <role rolename="hss_user"/>
        <role rolename="hss_admin"/>
        <user name="hss" password="hss" roles="hss_user"/>
        <user name="hssAdmin" password="hss" roles="hss_user,admin"/>
</tomcat-users>
```

C.6. AS: Presence

C.6.1. Fichero de configuración local del PRAS: /etc/kamailio/kamailio-local.cfg

```
#!define WITH_MYSQL
#!define WITH_PRESENCE
alias="example.org"
dns_udp_pref=1
dns_tcp_pref=1
dns_tls_pref=1
dns_sctp_pref=1
dns_try_naptr=on
loadmodule "enum.so"
```

C.6.2. Fichero de configuración para las herramientas de kamailio: /etc/kamailio/kamctlrc

```
SIP_DOMAIN=example.org
DBENGINE=MYSQL
```

C.6.3. Script de regeneración de la base de datos general para kamailio: /tmp/kam_pras_restore.sh

```
#!/usr/bin/expect -f
spawn mysql -u root -p -h localhost kamailio -Bse "source /tmp/kam_pras_restore.sql"
expect "Enter password:"
send "\r"
expect "# "
```

C.7. AS: Voicemail

C.7.1. Fichero de definición de la vista para el AS de voicemail en el HSS: /tmp/asterisk.sql

```
-- MySQL dump 10.13  Distrib 5.1.73, for debian-linux-gnu (i486)
--
-- Host: localhost    Database: asterisk
-- ------------------------------------------------------
```

```
-- Server version       5.1.73-1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

/*!40000 DROP DATABASE IF EXISTS `asterisk`*/;

CREATE DATABASE /*!32312 IF NOT EXISTS*/ `asterisk` /*!40100 DEFAULT CHARACTER SET utf8 */;

USE `asterisk`;

GRANT ALL ON asterisk.* to asterisk@203.0.113.6 IDENTIFIED BY 'asterisk';
--
-- Temporary table structure for view `sip`
--

DROP TABLE IF EXISTS `sip`;
/*!50001 DROP VIEW IF EXISTS `sip`*/;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
/*!50001 CREATE TABLE `sip` (
 `name` tinyint NOT NULL,
  `k` tinyint NOT NULL,
  `type` tinyint NOT NULL,
  `secret` tinyint NOT NULL,
  `host` tinyint NOT NULL,
  `callerid` tinyint NOT NULL,
  `context` tinyint NOT NULL,
  `mailbox` tinyint NOT NULL,
  `nat` tinyint NOT NULL,
  `qualify` tinyint NOT NULL,
  `fromuser` tinyint NOT NULL,
  `authuser` tinyint NOT NULL,
  `fromdomain` tinyint NOT NULL,
  `insecure` tinyint NOT NULL,
  `canreinvite` tinyint NOT NULL,
  `disallow` tinyint NOT NULL,
  `allow` tinyint NOT NULL,
  `restrictcid` tinyint NOT NULL,
  `ipaddr` tinyint NOT NULL,
  `port` tinyint NOT NULL,
  `regseconds` tinyint NOT NULL
) ENGINE=MyISAM */;
SET character_set_client = @saved_cs_client;

--
-- Temporary table structure for view `voicemail`
--

DROP TABLE IF EXISTS `voicemail`;
/*!50001 DROP VIEW IF EXISTS `voicemail`*/;
SET @saved_cs_client     = @@character_set_client;
SET character_set_client = utf8;
/*!50001 CREATE TABLE `voicemail` (
 `uniqueid` tinyint NOT NULL,
  `customer_id` tinyint NOT NULL,
  `context` tinyint NOT NULL,
  `mailbox` tinyint NOT NULL,
  `password` tinyint NOT NULL,
  `fullname` tinyint NOT NULL,
  `email` tinyint NOT NULL,
  `pager` tinyint NOT NULL,
  `stamp` tinyint NOT NULL
) ENGINE=MyISAM */;
SET character_set_client = @saved_cs_client;

--
-- Final view structure for view `sip`
--

/*!50001 DROP TABLE IF EXISTS `sip`*/;
/*!50001 DROP VIEW IF EXISTS `sip`*/;
/*!50001 SET @saved_cs_client          = @@character_set_client */;
/*!50001 SET @saved_cs_results         = @@character_set_results */;
/*!50001 SET @saved_col_connection     = @@collation_connection */;
/*!50001 SET character_set_client      = latin1 */;
/*!50001 SET character_set_results     = latin1 */;
/*!50001 SET collation_connection      = latin1_swedish_ci */;
/*!50001 CREATE ALGORITHM=UNDEFINED */
```

```
/*!50013 DEFINER=`root`@`localhost` SQL SECURITY DEFINER */
/*!50001 VIEW `sip` AS select `hss_db`.`impi`.`k` AS `name`,`hss_db`.`impi`.`k` AS `k`,\
'friend' AS `type`,NULL AS `secret`,'dynamic' AS `host`,`hss_db`.`impi`.`identity` AS `callerid`,\
'default' AS `context`,`hss_db`.`impi`.`k` AS `mailbox`,'no' AS `nat`,'no' AS `qualify`,\
 NULL AS `fromuser`,NULL AS `authuser`,NULL AS `fromdomain`,NULL AS `insecure`,'no' AS `canreinvite`,\
 NULL AS `disallow`,NULL AS `allow`,NULL AS `restrictcid`,NULL AS `ipaddr`,NULL AS `port`,\
 NULL AS `regseconds` from `hss_db`.`impi` */;
/*!50001 SET character_set_client      = @saved_cs_client */;
/*!50001 SET character_set_results     = @saved_cs_results */;
/*!50001 SET collation_connection      = @saved_col_connection */;

--
-- Final view structure for view `voicemail`
--

/*!50001 DROP TABLE IF EXISTS `voicemail`*/;
/*!50001 DROP VIEW IF EXISTS `voicemail`*/;
/*!50001 SET @saved_cs_client          = @@character_set_client */;
/*!50001 SET @saved_cs_results         = @@character_set_results */;
/*!50001 SET @saved_col_connection     = @@collation_connection */;
/*!50001 SET character_set_client      = latin1 */;
/*!50001 SET character_set_results     = latin1 */;
/*!50001 SET collation_connection      = latin1_swedish_ci */;
/*!50001 CREATE ALGORITHM=UNDEFINED */
/*!50013 DEFINER=`root`@`localhost` SQL SECURITY DEFINER */
/*!50001 VIEW `voicemail` AS select `hss_db`.`impi`.`id_imsu` AS `uniqueid`,\
`hss_db`.`impi`.`k` AS `customer_id`,'default' AS `context`,`hss_db`.`impi`.`k` AS `mailbox`,\
NULL AS `password`,`hss_db`.`impi`.`k` AS `fullname`,`hss_db`.`impi`.`identity` AS `email`,\
NULL AS `pager`,now() AS `stamp` from `hss_db`.`impi` */;
/*!50001 SET character_set_client      = @saved_cs_client */;
/*!50001 SET character_set_results     = @saved_cs_results */;
/*!50001 SET collation_connection      = @saved_col_connection */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2016-03-13 17:06:04
```

C.7.2. Fichero de configuración general de mysql en el HSS: /etc/mysql/my.cnf

```
[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock

[mysqld_safe]
socket          = /var/run/mysqld/mysqld.sock
nice            = 0

[mysqld]
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
language        = /usr/share/mysql/english
skip-external-locking
bind-address            = 0.0.0.0
key_buffer              = 16M
max_allowed_packet      = 16M
thread_stack            = 192K
thread_cache_size       = 8
myisam-recover         = BACKUP
query_cache_limit       = 1M
query_cache_size        = 16M
expire_logs_days        = 10
max_binlog_size         = 100M

[mysqldump]
quick
quote-names
max_allowed_packet      = 16M

[isamchk]
key_buffer              = 16M

!includedir /etc/mysql/conf.d/
```

C.7.3. Fichero de configuración sip.conf: /etc/asterisk/sip.conf

```
[general]
context=default         ; Default context for incoming calls
allowguest=no                   ; Allow or reject guest calls (default is yes, this can also be set to 'osp'
bindport=5060                   ; UDP Port to bind to (SIP standard port is 5060)
bindaddr=0.0.0.0                ; IP address to bind to (0.0.0.0 binds to all)
srvlookup=yes                   ; Enable DNS SRV lookups on outbound calls
domain=example.org              ; Set default domain for this host
domain=203.0.113.6                      ; Add IP address as local domain
allowexternalinvites=no ; Disable INVITE and REFER to non-local domains
;autodomain=yes                 ; Turn this on to have Asterisk add local host
pedantic=yes                    ; Enable slow, pedantic checking for Pingtel
;tos=184                        ; Set IP QoS to either a keyword or numeric val
;tos=lowdelay                   ; lowdelay,throughput,reliability,mincost,none
;maxexpiry=3600                 ; Max length of incoming registration we allow
;defaultexpiry=120              ; Default length of incoming/outoing registration
;notifymimetype=text/plain      ; Allow overriding of mime type in MWI NOTIFY
checkmwi=10                     ; Default time between mailbox checks for peers
vmexten=default                 ; dialplan extension to reach mailbox sets the
;videosupport=yes               ; Turn on support for SIP video
;recordhistory=yes              ; Record SIP history by default
disallow=all                    ; First disallow all codecs
allow=ulaw                      ; Allow codecs in order of preference
language=en                     ; Default language setting for all users/peers
;rtpholdtimeout=300             ; Terminate call if 300 seconds of no RTP activity
;trustrpid = no                 ; If Remote-Party-ID should be trusted
;sendrpid = yes                 ; If Remote-Party-ID should be sent
useragent=Asterisk PBX          ; Allows you to change the user agent string
;dtmfmode = rfc2833             ; Set default dtmfmode for sending DTMF. Default: rfc2833
                                ; Other options:
                                ; info : SIP INFO messages
                                ; inband : Inband audio (requires 64 kbit codec -alaw, ulaw)
                                ; auto : Use rfc2833 if offered, inband otherwise

#include sip_users.conf
```

C.7.4. Fichero de configuración sip_users.conf: /etc/asterisk/sip:users.conf

```
[pcscf]
type=friend
context=default
host=203.0.113.2
insecure=very

[scscf]
type=friend
context=default
host=203.0.113.3
insecure=very

[scscf2]
type=friend
context=default
host=203.0.113.9
insecure=very
```

C.8. DNS

C.8.1. Fichero de configuración general del Bind: /etc/bind/named.conf

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

C.8.2. Fichero de configuración de características generales del Bind: /etc/bind/named.conf.options

```
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        auth-nxdomain no;    # conform to RFC1035
        //listen-on-v6 { any; };
};

// Just in case we need some logs..
logging {
        channel config {
                file "/var/log/bind/config.log";
                severity dynamic;
        };
        channel secure {
                file "/var/log/bind/secure.log";
                severity dynamic;
                print-category yes;
                print-severity yes;
                print-time yes;
        };
        category security { secure; };
        category update { secure; };
};
```

C.8.3. Fichero de configuración específico para el escenario IMS del Bind: /etc/bind/named.conf.local

```
//
// Do any local configuration here
//
// MAJO Start
// Define a key to use for authenticated updates
key "rndc-key" {
        algorithm hmac-md5;
        secret "+GSPAUpjsopqlXyA9f0phA==";
};

// Define where the rndc listen, allowed hosts using the above key
controls {
        inet 127.0.0.1 port 953
                allow { 127.0.0.1; } keys { "rndc-key"; };
};
//MAJO End

// Consider adding the 1918 zones here, if they are not used in your
// organization
include "/etc/bind/zones.rfc1918";

// Zone definition for the scenario
zone "." {
        type master;
        file "/etc/bind/db.root";
        allow-update {key "rndc-key";};
};

zone "example.org" {
        type master;
        file "/etc/bind/db.example.org";
        allow-update {key "rndc-key";};
};

zone "example.com" {
        type master;
        file "/etc/bind/db.example.com";
        allow-update {key "rndc-key";};
};
```

```
zone "in-addr.arpa" {
        type master;
        file "/etc/bind/db.in-addr.arpa";
        allow-update {key "rndc-key";};

};

zone "113.0.203.in-addr.arpa" {
        type master;
        file "/etc/bind/db.203.0.113";
        allow-update {key "rndc-key";};

};

zone "100.51.198.in-addr.arpa" {
        type master;
        file "/etc/bind/db.198.51.100";
        allow-update {key "rndc-key";};

};

zone "1.e164.arpa" {
        type master;
        file "/etc/bind/db.1.e164.arpa";
        allow-update {key "rndc-key";};
};

zone "2.e164.arpa" {
        type master;
        file "/etc/bind/db.2.e164.arpa";
        allow-update {key "rndc-key";};
};
zone "2.nrenum.net" {
        type master;
        file "/etc/bind/db.2.nrenum.net";
        allow-update {key "rndc-key";};
};
zone "1.nrenum.net" {
        type master;
        file "/etc/bind/db.1.nrenum.net";
        allow-update {key "rndc-key";};
};
```

C.8.4. Fichero de configuración del rndc: /etc/bind/conf-rndc.conf

```
key "rndc-key" {
        algorithm hmac-md5;
        secret "+GSPAUpjsopqlXyA9f0phA==";
};

options {
        default-key "rndc-key";
        default-server 127.0.0.1;
        default-port 953;
};
```

C.8.5. Fichero de configuració de la clave del rdnc: /etc/bind/rndc.key

```
key "rndc-key" {
        algorithm hmac-md5;
        secret "+GSPAUpjsopqlXyA9f0phA==";
};
```

C.8.6. Mapa de la zona 113.0.203.in-addr.arpa: /etc/bind/db.203.0.113.in-addr.arpa

```
$TTL    604800
@       IN      SOA     ns1.example.org. hostmaster.example.org. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@               IN              NS      ns.example.org.
1               IN              PTR     ns.example.org.
2       IN      PTR     pcscf.example.org.
```

```
3       IN      PTR     scscf.example.org.
4       IN      PTR     icscf.example.org.
5       IN      PTR     hss.example.org.
6       IN      PTR     voicemail.example.org.
7               IN      PTR     alice.example.org.
8       IN      PTR     bob.example.org.
9       IN      PTR     scscf2.example.org.
10              IN              PTR             presence.example.org.
11              IN      PTR     claire.example.org.
12      IN      PTR     pcscf2.example.org.
```

C.8.7. Mapa de la zona example.org: /etc/bind/db.example.org

```
$ORIGIN example.org.
$TTL 1W
@                       1D IN SOA       localhost. root.localhost. (
                                        2006101001      ; serial
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum

                        1D IN NS        ns
ns                      1D IN A         203.0.113.1

pcscf                   1D IN A         203.0.113.2
_sip.pcscf              1D SRV 0 0 5060 pcscf
_sip._udp.pcscf         1D SRV 0 0 5060 pcscf
_sip._tcp.pcscf         1D SRV 0 0 5060 pcscf

example.org.       1D IN A         203.0.113.2
example.org.       1D IN NAPTR 10 50 "s" "SIP+D2U"      ""      _sip._udp
example.org.       1D IN NAPTR 20 50 "s" "SIP+D2T"      ""      _sip._tcp

scscf                   1D IN A         203.0.113.3
_sip.scscf              1D SRV 0 0 5060 scscf
_sip._udp.scscf         1D SRV 0 0 5060 scscf
_sip._tcp.scscf         1D SRV 0 0 5060 scscf

icscf                   1D IN A         203.0.113.4
_sip                    1D SRV 0 0 5060 icscf
_sip._udp               1D SRV 0 0 5060 icscf
_sip._tcp               1D SRV 0 0 5060 icscf

hss                     1D IN A         203.0.113.5
voicemail               1D IN A         203.0.113.6
voice                   1D IN CNAME     voicemail
alice                   1D IN A         203.0.113.7
bob                     1D IN A         203.0.113.8

scscf2                   1D IN A         203.0.113.9
_sip.scscf2              1D SRV 0 0 5060 scscf2
_sip._udp.scscf2         1D SRV 0 0 5060 scscf2
_sip._tcp.scscf2         1D SRV 0 0 5060 scscf2

presence                1D IN A         203.0.113.10
pras                    1D IN CNAME     presence

claire                  1D IN A         203.0.113.11

pcscf2                   1D IN A         203.0.113.12
_sip.pcscf2              1D SRV 0 0 5060 pcscf2
_sip._udp.pcscf2         1D SRV 0 0 5060 pcscf2
_sip._tcp.pcscf2         1D SRV 0 0 5060 pcscf2
```

Bibliografía

[1] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588 (Proposed Standard), September 2003. Obsoleted by RFC 6733, updated by RFCs 5729, 5719, 6408.

[2] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[3] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782 (Proposed Standard), February 2000. Updated by RFC 6335.

[4] M. Mealling. Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database. RFC 3403 (Proposed Standard), October 2002.

[5] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

[6] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.

[7] J. Rosenberg and H. Schulzrinne. Session Initiation Protocol (SIP): Locating SIP Servers. RFC 3263 (Proposed Standard), June 2002.

[8] J. Rosenberg and H. Schulzrinne. An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing. RFC 3581 (Proposed Standard), August 2003.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878, 7462, 7463.

[10] H. Schulzrinne and S. Petrack. RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals. RFC 2833 (Proposed Standard), May 2000. Obsoleted by RFCs 4733, 4734.