Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema de evaluación de proveedores telefónicos

Telephony provider evaluation system

Autor

Noel Mateo Comín

Director

Fernando Orús Morlans

Ponente

Julián Fernández Navajas

Escuela de Ingeniería y Arquitectura

2019

Repositorio de la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

Sistema de evaluación de proveedores telefónicos

Resumen

En los sistemas telefónicos y en particular en la telefonía IP (donde la comunicación de interlocutores es a través de un sistema que mantiene las funcionalidades de la telefonía tradicional pero que permite el transporte de la voz empaquetada en datagramas IP), sobre todo en llamadas internacionales, la comunicación entre interlocutores no depende únicamente de una operadora, sino de un conjunto de proveedores (*Carrier*) que se "traspasan" la llamada hasta dar con su destinatario (la operadora del destinatario móvil). La operadora final termina la llamada entregándola a su cliente.

Durante este Trabajo Fin de Grado se ha desarrollado un sistema que permite la evaluación de proveedores de telefonía IP (Tanto *Carrier* como Operadoras), como solución al desconocimiento de comportamientos que toman las llamadas que son encaminadas a través de sus redes.

Tanto el análisis previo como un estudio de las tecnologías a utilizar han sido aplicados en el desarrollo de un sistema que permite la sincronización sobre un servidor central de la información de llamadas realizadas por un generador y recibidas por terminales móviles. Para ello, se ha desarrollado una aplicación para programar de manera automatizada tanto la realización de llamadas (utilizando un generador de llamadas proporcionado por la empresa ***Business Telecommunication Services*** con el objetivo de integrarlo a nuestro sistema) como la recolección de datos de las mismas, una aplicación Android que instalada en terminales móviles recoge y envía los datos de llamadas que le ha realizado nuestro generador, y finalmente, una aplicación de tipo *Rest* (*Representational State Transfer*) que recibe los datos de las llamadas que nos envía el terminal Android, guardándolos en la base de datos del servidor.

Se ha comprobado la flexibilidad del sistema instalándolo sobre diferentes dispositivos y ubicaciones, y se ha verificado su funcionamiento mediante la realización de pruebas sobre un escenario real.

Telephony provider evaluation system

Abstract

In the telephonic systems, specially in the IP telephony (where the communication between interlocutors is conducted through a system that keeps the functionality of the traditional telephony but allows the transport of packed voice in IP datagrams), above all in international calls, the communication between interlocutors , doesn't depends on one single operator, but a group of providers (Carrier) who transfer the call until they reach the destination (the target operator). The final operator, ends the call by giving it to its client.

During this Final Degree Project, we have developed a system that allows the providers evaluation(Both Carriers and Operators) , as well as the solution to the unknowledge of the behaviors that the calls take.

Both the previous analisis and a study of the technologies that we are going to use, they have been applied in the developement of a system that allows the synchronization over a central server of the information of the calls made by a generator and received by mobile devices. For it, we have developed an app to program automatically both the calls launching (using a calls generator provided by the enterprise ***Business Telecommunication Services***, with the aim of integrate its functionality to our system) and the data collection , an Android app which installed in a mobile collects and sends the data of the calls that have been made by our generator , and finally , a Rest Aplication (Representational State Transfer) which receives the data sended by the Android terminal (mobile) , and save them in a server data base.

We have verify the system flexibility, by installing it in many differents devices and locations. Also, we have tested its proper functioning by doing some test in a real scenario.

Índice

Capítulo 1: Introducción

1.1 Introducción

Las comunicaciones telefónicas se han convertido desde su creación en una pieza clave a nivel global para el intercambio de información. Aunque las necesidades del ser humano nos han llevado a la creación de otros medios de comunicación, en un mundo cambiante tanto en el plano económico como el social siempre se recurre al medio telefónico cuando se requiere un mayor *feedback* en la comunicación.

Los avances tecnológicos nos han permitido que muchas de las llamadas que anteriormente se realizaban sobre la red telefonía convencional se realicen actualmente sobre telefonía IP, a través de Internet. Las ventajas de este tipo de telefonía con respecto a la telefonía convencional (sobre todo a nivel de coste) hacen que su utilización crezca cada vez más, sobre todo a nivel empresarial.

La necesidad de las Operadoras de telefonía IP de conectar las llamadas telefónicas de sus clientes con sus destinatarios, las lleva a comunicarse con propietarios de red troncales denominados *Carrier*. Estos a su vez se comunican con otros *Carrier*, con el objetivo de encaminar la llamada hasta la Operadora final. Finalmente, la operadora entregara la llamada a su destinatario.

Tal y como ya podremos presuponer, el problema de este tipo de telefonía reside en el mantenimiento de la calidad de la comunicación, ya que la dependencia en intermediarios para entregar la llamada puede dar lugar a una disminución en la calidad del servicio.

Algunos proveedores de red, con el objetivo de maximizar sus beneficios, derivan las llamadas a otras redes que no garantizan buenas calidades de servicio, dando lugar a situaciones que incumplen las calidades garantizadas a los clientes. En algunos casos, se dan cambios de número llamante por parte de los proveedores, en otros casos los proveedores facturan incorrectamente las llamadas, sumando segundos a su duración o incluso dando por descolgada la llamada aunque el cliente no termine de comunicarse con su destinatario (*False Answer Supervision*).

En definitiva, la calidad del servicio dada por un proveedor depende tanto de sí mismo como de los intermediarios en el camino de la llamada. Por ello, es necesario conocer los comportamientos de los diferentes *Carrier* y Operadoras para saber si podemos efectuar llamadas a través de sus redes.

1.2 Motivación y objetivos

Partiendo del problema de calidad de servicio descrito en la introducción, la idea de este Trabajo Fin de Grado surge de las necesidades de dar un servicio de calidad a sus clientes por parte de la empresa BTS (*Business Telecommunication Services*).

BTS es un *Carrier* telefónico con licencia en los EEUU, América Latina, Europa y África, cuya fuente principal de ingresos deriva de conectar llamadas telefónicas de sus clientes con proveedores (*Carrier* u otras Operadoras). Clasificado en el Top 15 en el mercado mayorista, su red gestiona más de 8 mil millones de minutos de llamadas a nivel mundial, bajo eso sí, cierta calidad de servicio exigida por sus clientes.

Tal y como se ha comentado en la introducción, para poder conectar llamadas telefónicas es necesario encaminar tráfico telefónico a través de redes cuyos propietarios pueden respetar o no las calidades de servicio mínimas exigidas por BTS. Debido a la cantidad de minutos que se manejan, derivar llamadas por redes equivocadas podría desembocar no solo en pérdidas de beneficios, sino en quejas de clientes por una disminución en la calidad de servicio. Un problema que no depende de ellos directamente y que por lo tanto conviene analizar.

Por ello, el objetivo de este trabajo fin de grado es el desarrollo de un sistema de evaluación de proveedores con el fin de detectar este tipo de anomalías. Para ello, se sincronizará y comparará en un servidor la información de las llamadas realizadas por un equipo proporcionado por BTS (generador de llamadas) y recibidas por terminales móviles Android.

1.3 Organización de la memoria

La memoria se ha organizado con la siguiente disposición:

- Capítulo 1: se presenta la descripción del trabajo, la problemática del mismo y los objetivos a alcanzar.
- Capítulo 2: se repasa el estado del arte, y se explica el análisis que se ha realizado previo al desarrollo.
- Capítulo 3: se explica el desarrollo y funcionamiento del sistema de aplicaciones.
- Capítulo 4: se detalla el despliegue realizado del sistema sobre diferentes ubicaciones.
- Capítulo 5: explicamos las pruebas realizadas durante el trabajo y mostramos los resultados de las mismas.
- Capítulo 6: se exponen las conclusiones finales del trabajo y las posibles líneas futuras.

Finalmente, se incluyen anexos al final del documento:

- Red de la empresa
- Ejemplo de funcionamiento: *Telecop*
- Ejemplo de funcionamiento: Aplicación Android
- Nomenclatura
- Códigos *Release Cause*
- Guía de instalación: Servidor
- Guía de usuario: Servidor
- Guía de instalación y uso: Aplicación Android
- Planificación temporal

Capítulo 2: Análisis Previo

2.1 Herramientas

A continuación, se enumeran las herramientas, lenguajes de programación y plataformas utilizadas para el desarrollo de este trabajo:

- **Android Studio:** Plataforma proporcionada por Google para el desarrollo de aplicaciones Android. Permite realizar virtualizaciones de terminales móviles para la prueba de aplicaciones en desarrollo. Soporta los lenguajes *Kotlin* y *Java*. Se utiliza en el desarrollo de la aplicación Android.
- **Spring Framework:** Plataforma de código abierto que nos permite desarrollar aplicaciones web (entre otras) de manera rápida y eficaz, ahorrando código y por tanto reduciendo el tiempo de desarrollo. Soporta el lenguaje *Java*. Se utiliza en el desarrollo de las aplicaciones del servidor.
- **Wireshark:** Software de análisis de capturas de red. Se trata de una aplicación con interfaz gráfica con la que visualizar los datos capturados e interpretarlos de una manera más amigable.
- **Postman:** aplicación gráfica para la realización de peticiones *http* de diferente tipo, permitiendo interactuar con *APIs Rest*. Utilizado para la realización de pruebas con la aplicación *Rest*.
- **PostgreSQL:** sistema de gestión de bases de datos relacional, orientado a objetos y de código abierto. Permite la configuración, creación y manejo de varias bases de datos sobre la máquina que se instala.
- **Pgadmin:** aplicación para gestionar gráficamente *PostgreSQL*.
- **OpenVPN:** software *opensource* que permite la conexión segura de recursos locales desde ubicaciones remotas. Utilizado para la conexión con la base de datos de BTS.
- **GitLab**: Servicio web de control de versiones y desarrollo colaborativo. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una licencia de código abierto.
- **Java**: Lenguaje de programación de propósito general, orientado a objetos muy extendido y con escasas dependencias de implementación. Hace uso de una máquina virtual java (JVM) para ejecutarse, puede ejecutarse en cualquier arquitectura. Se utiliza en el desarrollo de las aplicaciones de cliente y servidor.

2.2 Escenario Previo

El escenario previo al desarrollo del sistema consta de terminales con cobertura y conexión a internet, de la infraestructura de red que tienen las Operadoras y *Carrier* de telefonía para conectar las llamadas, y de la red que tiene BTS tanto para conectar las llamadas como para realizarlas.

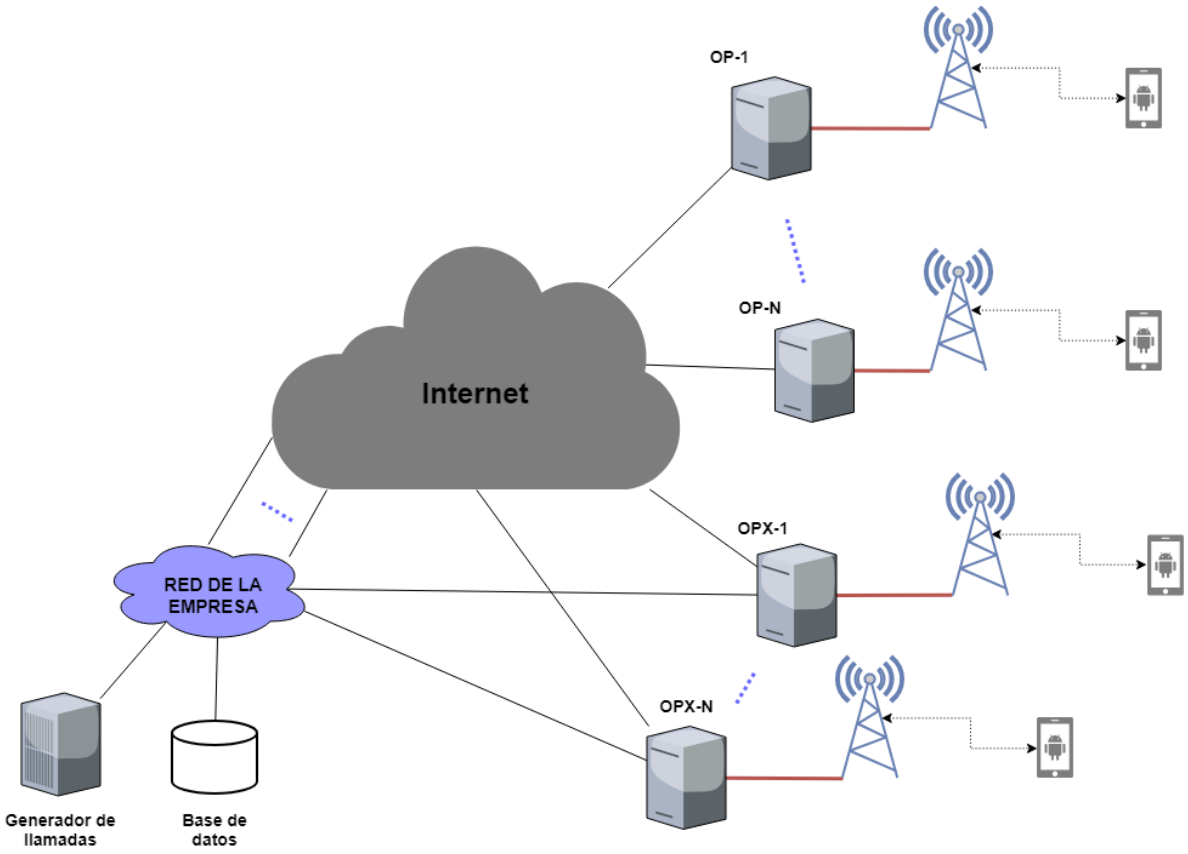


Figura 1: Escenario Previo

De la red de la empresa, explicada en el anexo "**Red de la empresa**", penden los generadores de llamadas utilizados para realizar y conectar llamadas IP a través de los proveedores con los que se trabaje. Los proveedores (*Carrier* u Operadoras) representados mediante OP son aquellos con los que la empresa trabaja a través de internet, y los representados mediante OPX son con los que se ha establecido un enlace dedicado, y por tanto la llamada pasa de la red de BTS a la red del proveedor de forma directa, sin derivar la llamada por internet.

Cuando una llamada llega a uno de los proveedores, este se encargara de finalizar la llamada de la manera que le sea posible. Si pueden entregar la llamada a su cliente final lo harán (comportamiento de operadora), sino reenviarán la llamada a otro proveedor hasta que la llamada se conecte con su destinatario (comportamiento de *Carrier*). En caso de que los proveedores no encuentren su destinatario móvil la llamada no será conectada.

Las llamadas realizadas por un generador quedarán registradas en las bases de datos de la empresa. Así, en caso de necesitarlo, tendremos la información lo mejor estructurada posible.

En el desarrollo de este trabajo fin de grado era necesario conocer cómo utilizar uno de los generadores de llamadas de BTS. Para ello, se procedió al estudio de *Telecop*.

Telecop es una aplicación desarrollada por la empresa que entre otras cosas utiliza *FreeSwitch* (aplicación de software libre que representa un generador de llamadas) para realizar llamadas.

Como concepto general, la manera que tiene *Telecop* de programar llamadas se resume en escribir la hora, el número llamado, el número llamante y el proveedor en una tabla de su base de datos. Posteriormente, un script recogerá los datos de dicha tabla y programará las llamadas en *FreeSwitch*.

Como concepto específico debemos saber que la base de datos de *Telecop* contiene infinidad de tablas, pero las que nos interesan para programar llamadas son las tablas *probe_call* y *tb_callbatteries*.

La tabla *tb_callbatteries* nos permite definir el tipo de llamadas que vamos a realizar (periódicas o no), y a través de qué continente o continentes se van a realizar las llamadas.

- **Id:** Primary key. identificador de la fila que definimos
- **Type:** tipos de llamada. Existen los tipos Periódicos o ONE_OFF
- **Continents_id:** identificador de los continentes por los que se realizaran llamadas

Es necesario tener rellenada una fila de esta tabla para poder rellenar la tabla *probe_call* con tantas filas como queramos.

La tabla *probe_call* contiene toda la información necesaria para la realización de llamadas:

- **Ani**: número de teléfono con el que realizar la llamada (número llamante).
- **Area_code**: código de área del proveedor. Cada proveedor puede tener varios códigos de área.
- **Call_battery_id**: *foreign key*. Id de la tabla *tb_callbatteries.*
- **Called_number**: número llamado.
- **Destination**: columna redundante equivalente a *area_code.*
- **Resource_group**: identificador del proveedor.
- **Status**: estado de la programación de la llamada (*ok, off, scheduled*).
- **Probe_call_idx**: sin uso.
- **Unix_time**: hora programada para la realización de la llamada en segundos.
- **Db_time**: hora y fecha equivalente a *Unix_time*.

Para generar una llamada únicamente debemos rellenar esta tabla introduciendo los campos correctamente, y poniendo la columna status a ***scheduled***. El script anteriormente mencionado revisará la tabla cada **15 minutos** y si dicha columna esta a ***scheduled*** recogerá la fila entera, programando la llamada en *FreeSwitch*.

Cuando la llamada se realice, *FreeSwitch* escribirá en su fichero ***log*** información sobre el resultado de cada una de las llamadas que ha realizado (hora, duración, proveedor, número llamado,…). Como trabajar con dicho log es tedioso existe otro script que **cada minuto dos de cada hora** coge la información del mismo y la pasa a la tabla ***bts_fs_cdr*** de la base de datos de *Telecop*.

A continuación, se enumera la información de relevancia de la tabla *bts_fs_cdr*:

- **Charged_time**: duración de la llamada conectada.
- **Called_number**: número llamado.
- **Call_número**: número utilizado como llamante.
- **Resource_group_in**: identificador del proveedor por el que se ha enviado la llamada.
- **Release_cause**: identificador de la causa de liberación de la llamada.

Como resultado del estudio, hemos concluido que para realizar llamadas debemos insertar filas en la tabla *probe_call*, y para recoger el resultado de las mismas debemos leer la tabla *bts_fs_cdr*. Eso sí, teniendo en cuenta los periodos de tiempo en los que lo scripts recogen los datos.

En el anexo “Ejemplo de funcionamiento: Telecop” se muestra un ejemplo del proceso que sigue *Telecop* para realizar una llamada y recoger su resultado.

Capítulo 3: Sistema desarrollado

3.1 Planteamiento General

El firme objetivo de comparar información entre llamadas realizadas por *FreeSwitch* y recibidas por terminales móviles, nos ha llevado a desarrollar un sistema que aproveche la infraestructura de BTS para realizar llamadas, y que sincronice la información del resultado de las mismas. Para ello, sobre la red planteada en la "Figura 1: Escenario Previo", se ha instalado un servidor sobre una red LAN con conexión a internet.

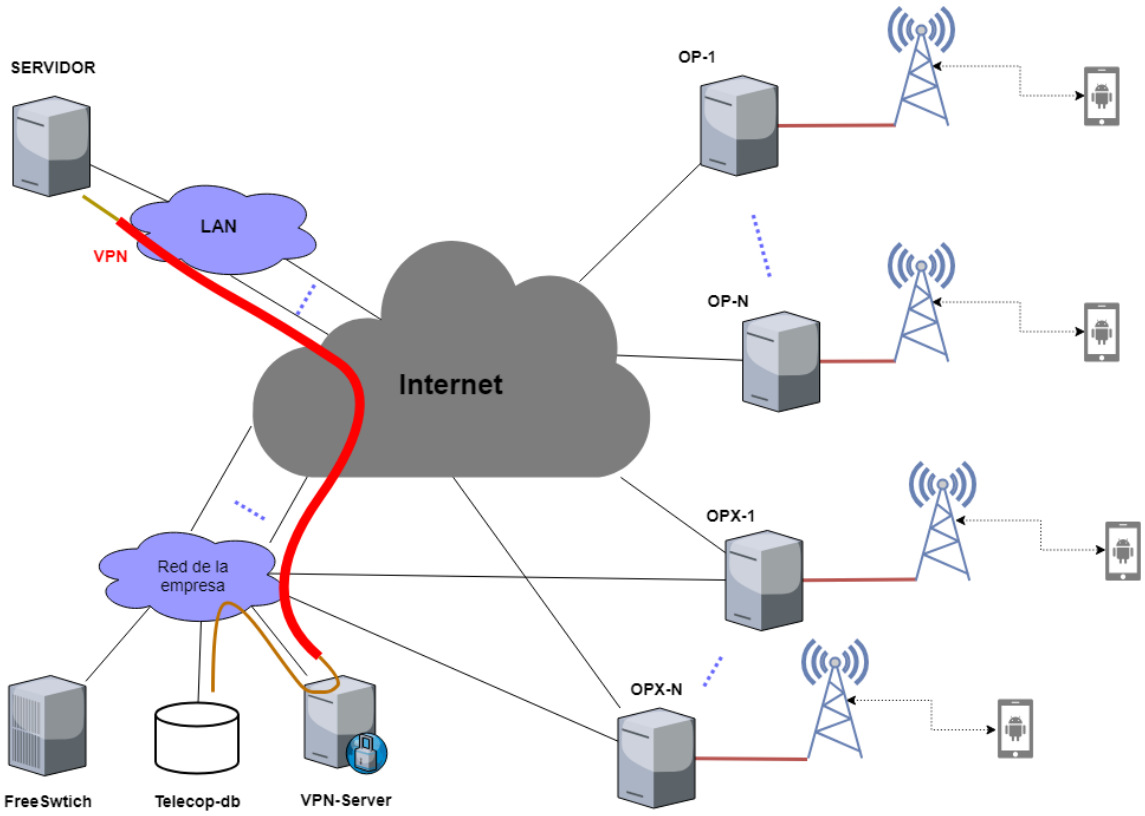


Figura 2: Escenario de desarrollo

El servidor será el nexo de unión del sistema recibiendo la información tanto de llamadas realizadas por *FreeSwitch* como de llamadas recibidas por los terminales.

Para que desde el servidor se puedan tanto programar llamadas como recoger sus resultados es necesaria la configuración de una VPN (*Virtual Private Network*), permitiendo así tanto la lectura como escritura en la base de datos de *Telecop*.

Sobre la estructura que se ha planteado, se han instalado las aplicaciones desarrolladas tanto sobre el servidor como sobre terminales móviles, relatando el funcionamiento conjunto del sistema a continuación:

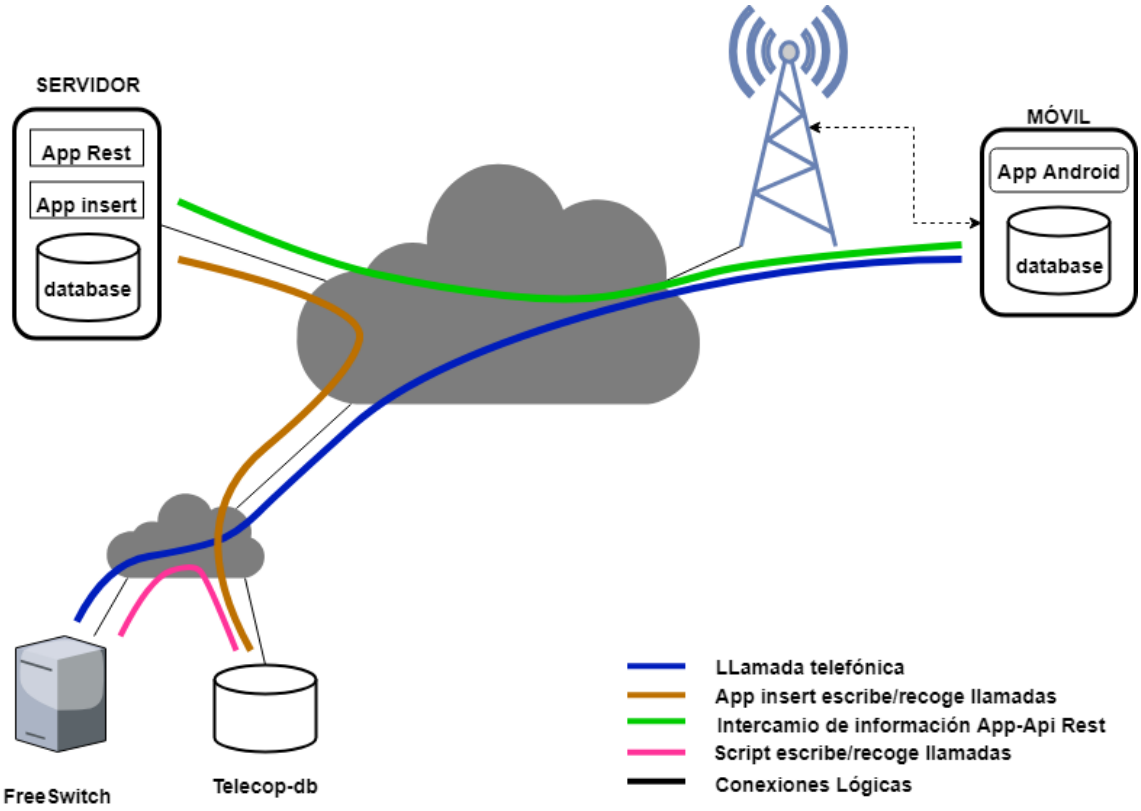


Figura 3: Diagrama de comunicaciones entre equipos

En primer lugar, para poder comparar información de llamadas es necesario realizarlas. Para ello, se ha establecido un fichero ubicado en el servidor sobre el que escribiremos los datos necesarios para la programación de las mismas.

La aplicación encargada (*App insert*), leerá dicho fichero y programará las llamadas escribiendo en la tabla *probe_call* de la base de datos de *Telecop* (en marrón). Un script proporcionado por la empresa recogerá los datos de dicha tabla y programará finalmente las llamadas en *FreeSwitch* (rosa).

Cuando la llamada haya sido realizada otro script recogerá los resultados de la llamada obtenidos por *FreeSwitch* (duración, hora, número de teléfono…), y los guardará en la tabla *bts_fs_cdr* de la base de datos de *Telecop*. A continuación, la aplicación *App insert* recogerá los resultados de la base de datos de *Telecop*, y los guardará en la base de datos de nuestro servidor (marrón).

Llegados a este punto, tenemos en nuestro servidor la información del resultado de la llamada por parte de *FreeSwitch*, por lo que únicamente nos falta la información por parte del terminal Android.

Cuando desde la aplicación Android se realice un *Report* (en verde), se pedirá al servidor información de llamadas que le han sido realizadas (hora de la llamada, duración, teléfono llamante...). El servidor leerá la petición del móvil a través de su interfaz *Rest* (*Aplicación Rest*) y le enviará una respuesta con los datos de dichas llamadas (verde).

El móvil recibirá la información y buscará en su registro las llamadas recibidas en las horas que el servidor le ha llamado. Cogerá el resultado (hora de la llamada, duración, teléfono llamante...) y lo guardará en su base de datos local, enviando de vuelta la información al servidor cuando le sea posible.

Por su parte, el servidor recibirá los datos que le envía el terminal guardándolos en su base de datos, obteniendo así en una tabla el resultado de la llamada tanto por parte del terminal Android como por parte de *FreeSwitch*.

3.2 Base de datos

Tal y como hemos visto en el planteamiento general, para poder tratar la información tanto de los terminales móviles como de *FreeSwitch* es necesario guardarla. Para ello, se ha creado una base de datos con PostgreSQL [3] en la dirección de *Localhost* y el puerto 5432 del servidor, denominada BTS.

La base de datos se ha creado sobre la ubicación en la que se están ejecutando las aplicaciones del servidor, pero igualmente se podría haber establecido sobre una ubicación remota.

A continuación, se enumeran las tablas creadas:

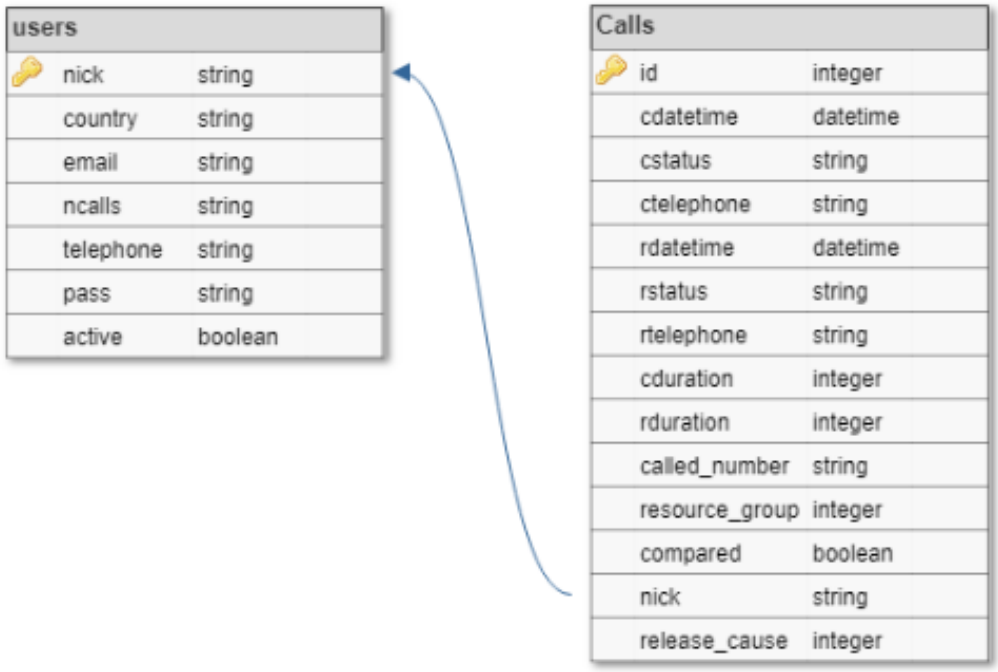


Figura 4: Diagrama relacional tabla *users* y *calls*

La tabla *users* tiene como clave primaria el *nick* del usuario que se ha registrado en la aplicación. Esta tabla contiene toda la información necesaria de los usuarios que la conforman. A continuación, se detallan los parámetros de la tabla:

- **Nick:** clave primaria de la tabla y *Nick* del usuario registrado.
- **Country:** país del usuario.
- **Email:** correo del usuario.

- **Ncalls:** número de llamadas que ha recibido el usuario.
- **Telephone:** número de teléfono del usuario.
- **Pass:** contraseña del usuario.
- **Active:** indica si el usuario ha sido activado.

Tanto el parámetro *ncalls* como el parámetro *active* no son utilizados en la actualidad, pero si serán útiles en el futuro de la aplicación.

La tabla ***Calls*** contiene información del resultado de las llamadas tanto por parte de *FreeSwitch* como por parte de los terminales Android. A continuación, se detallan los parámetros de la tabla:

- **Id:** clave primaria de la tabla que se incrementa automáticamente.
- **Cdatetime:** día y hora de la llamada realizada por parte de *Freeswitch*.
- **Cstatus:** información del estado de la llamada por parte de *Freeswitch* (*offhook*, *missed*, *rejected*, *notconnected*).
- **Ctelephone:** número con el que *Freeswitch* ha realizado la llamada.
- **Cduration:** información de la duración de la llamada por parte de *Freeswitch*.
- **Rdatetime:** día y hora de la llamada recibida por parte del terminal.
- **Rstatus:** información del estado de la llamada por parte del terminal (*offhook*, *missed*, *rejected*, *notfound*).
- **Rtelephone:** número que recibe el terminal.
- **Rduration:** información de la duración de la llamada por parte del terminal.
- **Called_numer:** número al que se ha realizado la llamada.
- **Resource_group:** identificador del proveedor por el que se ha realizado la llamada.
- **Nick:** *Foreign key* que relaciona la tabla *users* con la tabla *calls.*
- **Compared:** booleano que indica si la llamada ha sido contrastada por el terminal.
- **Release_cause:** identificador que representa información del estado concreto de la llamada (visualización de los códigos en el anexo “Códigos Release Cause”).

La tabla *PrivateProbeCall* es un duplicado local de tabla *probe_call* de la base de datos de *Telecop* y contiene información de llamadas que van a ser programadas en *FreeSwitch*.

Figura 5: Diagrama que representa la tabla *PrivateProbeCall*

- **Id:** identificador de la fila autogenerado.
- **Ani:** número de teléfono con el que realizar la llamada (número llamante).
- **Area_code:** código de área del proveedor.
- **Call_battery_id:** *foreign key*. Id de la tabla *tbs_callbattery.*
- **Called_number:** número llamado.
- **Destination:** columna redundante equivalente a *area_code.*
- **Resource_group:** identificador del proveedor.
- **Status:** estado de la programación de la llamada.
- **Probe_call_idx:** sin uso.
- **Unix_time:** hora programada para la realización de la llamada en segundos.
- **Db_time:** hora y fecha equivalente a *Unix_time.*

3.3 Aplicación *Rest*

3.3.1 Investigación Previa

En un sistema del que vamos a extraer información de terminales Android repartidos por el mundo es vital una correcta interacción cliente-servidor para el intercambio de información.

En la actualidad, la mayoría de las aplicaciones móviles intercambian información con servidores a través de una interfaz de tipo *Rest* propuesta por el proveedor de la aplicación. Un A*pi Rest* no es más que una interfaz para obtener o guardar datos de un sistema a través de peticiones *http* de distinto tipo (*get, post, put* y *delete*), aprovechando la infraestructura de *http* y ahorrando la apertura de puertos innecesarios para la realización de conexiones TCP.

Aprovechar la lógica *Rest* nos proporciona la escalabilidad necesaria para la realización de este proyecto ya que crecerá en busca de diferentes objetivos por parte de la empresa.

El desarrollo de esta aplicación se realiza con *Spring* Referencias

[1] y el lenguaje de modelado de información utilizado para las peticiones *http* es *JSON*.

3.3.2 Requisitos

3.3.2.1 Requisitos funcionales:

- **RF1:** La aplicación debe ser capaz de enviar/recibir información.
- **RF2:** La aplicación debe permitir guardar/extraer información.

3.3.2.2 Restricciones:

- La aplicación debe estar realizada en Java.

3.3.2.3 Diccionario de datos:

- Información: datos referidos al registro de los usuarios o datos referidos a información de las llamadas.
- Guardar/Extraer: El sistema debe permitir conectarse a una base de datos y guardar y extraer la información requerida.
- Enviar/Recibir: El sistema envía o recibe información a través de peticiones *http get* o *http post*.

3.3.3 Funcionamiento

La aplicación *Rest* está configurada para recibir peticiones *http* a través de cuatro *URL* diferentes, cada una con una funcionalidad:

- Registro
- Identificación
- Entregar llamadas
- Guardar llamadas

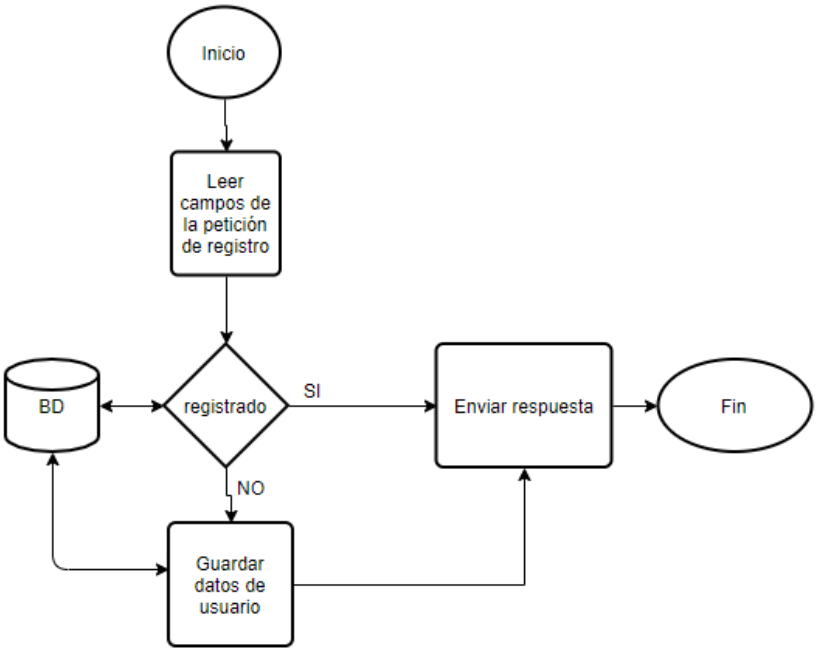


Figura 6: Diagrama de flujo de una petición http a la *url* de registro

Realizando una petición *http* de tipo *post* con información referida al registro de un usuario a la *url* http://ipserver:puerto/user se registrará un usuario en el sistema si no se ha registrado ya, guardándolo en la tabla ***users*** de la base de datos local. En caso de que el cliente ya haya sido registrado el servidor le devolverá un error.

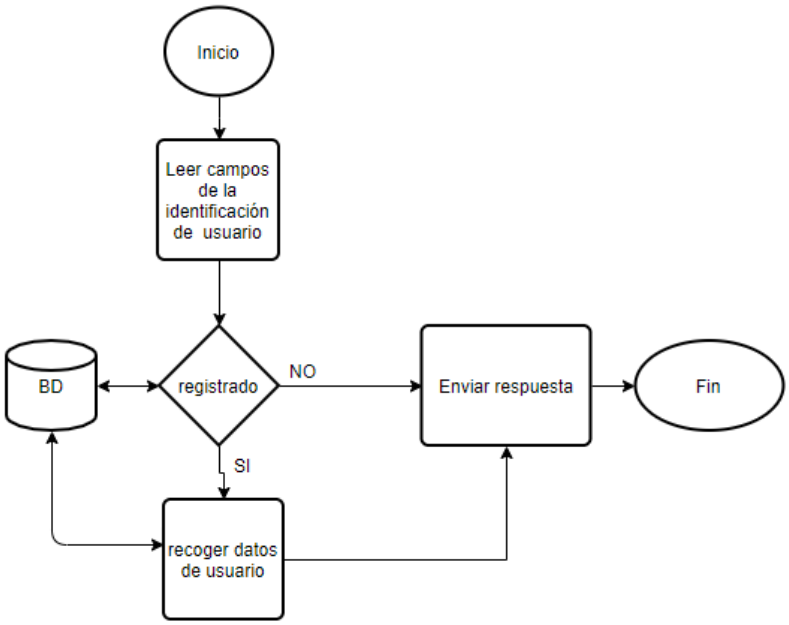


Figura 7: Diagrama de flujo de una petición *http* a la *url* de identificación

Si realizamos una petición *http* de tipo *post* a la *url* http://ipserver:puerto/id con los campos ***Nick*** y ***pass*** de un usuario registrado rellenados correctamente devolverá todos los datos del usuario al cliente, en caso contrario, el servidor le reportará un error.

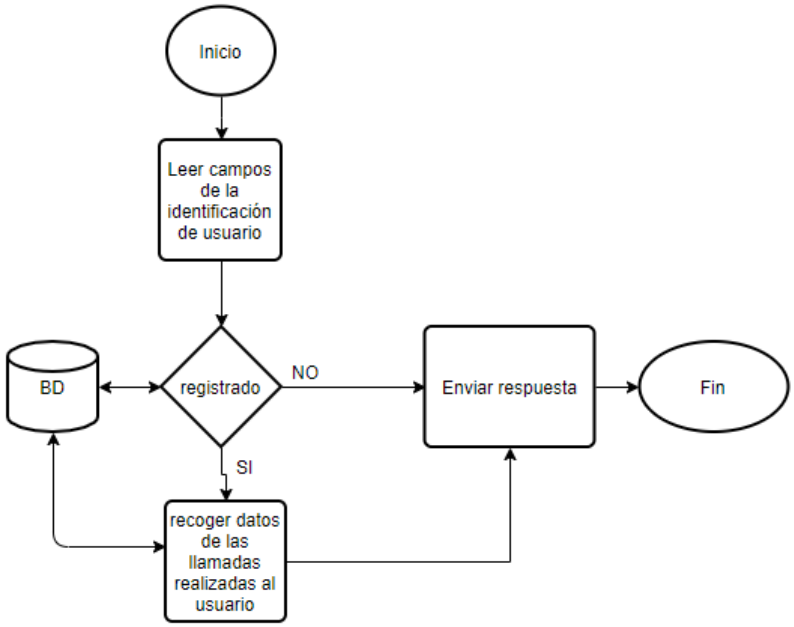


Figura 8: Diagrama de flujo de una petición *http* a la *url* de petición de llamadas

El usuario realizará peticiones http de tipo *get* a la *url* http://ipserver:puerto/getcalls/{nick} (*Nick* del usuario registrado) cuando quiera que el sistema le entregue la lista de las llamadas que el *FreeSwitch* le ha realizado. El sistema buscará las llamadas (filas) en la tabla ***calls*** que no han sido comparadas por el cliente, es decir, con el campo *compared* a *false* y se las enviará. En caso de que el usuario no esté registrado el servidor reportará un error.

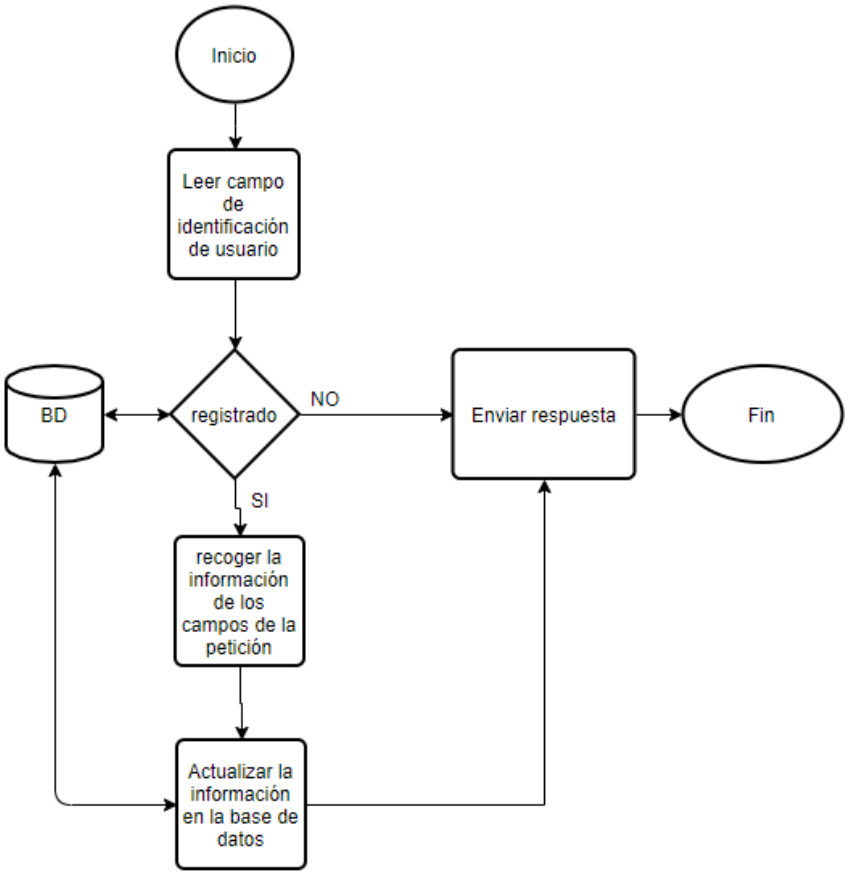


Figura 9: Diagrama de flujo de una petición *http* a la *url* de guardar llamadas

El usuario realizará una petición *http* de tipo *post* a la *url* http://ipserver:puerto/setcalls/{nick} con la información de las llamadas que ha recibido por parte de nuestro sistema. El servidor comprueba que el *Nick* es de un usuario registrado y actualiza la tabla ***calls*** de la base de datos en caso de que el *Nick* sea correcto.

3.4 Aplicación de inserción de llamadas

3.4.1 Investigación Previa

El hecho de tener que integrar el sistema de generación de llamadas de BTS con la posibilidad de insertar llamadas a nivel de usuario, nos hace investigar el funcionamiento de generación de llamadas proporcionado por la empresa (explicado concretamente en el "Escenario Previo").

Como resumen del punto "Escenario previo", para poder "crear" llamadas debemos insertar filas en la Tabla *ProbeCall* de la Base de datos de *Telecop*. Cada 15 minutos serán recogidas las nuevas filas que hayamos insertado en dicha tabla, programando las llamadas en *FreeSwtich*. Finalmente, las llamadas realizadas por *FreeSwitch* entre las N-1 y N horas son registradas en la base de datos de *Telecop* a las (*N horas+2min*), concretamente en la tabla *bts_fs_cdr*.

Para el desarrollo de esta aplicación se ha utilizado *Spring*, aprovechando las funcionalidades implícitas de dicha herramienta, como la ejecución de código de forma planificada.

3.4.2 Requisitos

3.4.2.1 Requisitos funcionales.

- La aplicación debe permitir insertar llamadas
- La aplicación debe permitir guardar llamadas
- La aplicación debe permitir recoger llamadas.

3.4.2.2 Restricciones

- La aplicación debe estar realizada en Java.

3.4.2.3 Diccionario de datos:

- **Llamadas:** una llamada se identifica con un número llamado, un número llamante, una hora y fecha, un proveedor y un código de área.
- **Guardar:** El sistema debe permitir conectarse a una base de datos y guardar la información requerida.
- **Recoger:** El sistema periódicamente recogerá información de los resultados de la llamada por parte de *FreeSwitch*.
- **Insertar:** la aplicación agregará las llamadas leídas de un fichero a la base de datos.

3.4.3 Funcionamiento

La funcionalidad de esta aplicación permite que un usuario del sistema realice llamadas de manera programada y que se recojan los resultados de las mismas para posteriormente compararlos con los de los terminales móviles.

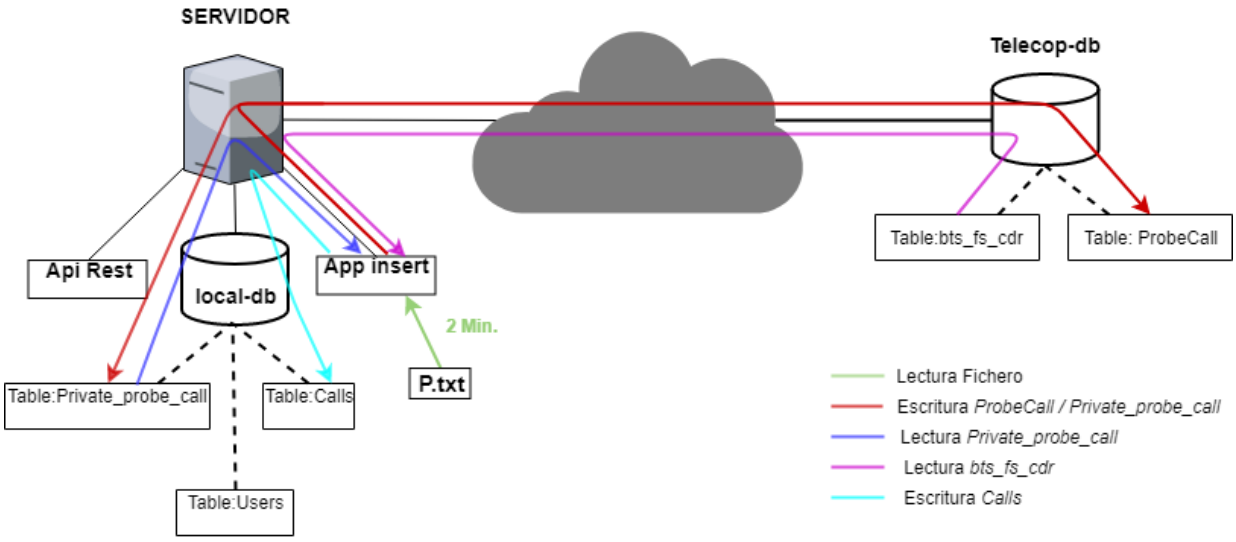


Figura 10: Diagrama del proceso de escritura/lectura de tablas

La manera que se ha definido en el desarrollo de esta aplicación para generar llamadas es mediante la escritura de filas (llamadas) en un fichero, de acuerdo al anexo "Guía de usuario: Servidor".

Periódicamente, esta aplicación recogerá los datos de un fichero denominado "p.txt" (verde) insertándolos en dos tablas (rojo). Por un lado los inserta en la tabla ***ProbeCall*** y por otro lado en la tabla ***PrivateProbeCall***, teniendo los datos de la llamada replicados en dos lugares diferentes.

La llamada se produce a la hora que hemos programado en *ProbeCall*, procediendo a la lectura del resultado de la misma accediendo a la tabla *bts_fs_cdr* a las **"N horas + 10 minutos"** en busca de las llamadas que se han realizado entre las horas **"N-1 y N"**.

La tabla *bts_fs_cdr* tiene datos de todas las llamadas que ha realizado la empresa, por lo tanto, para buscar únicamente las llamadas que hemos realizado nosotros debemos leer en la tabla *PrivateProbeCall* las llamadas que habíamos programado para realizarse entre las *N-1 y N* (morado), y compararlas con las que ha realizado toda la empresa en la tabla *bts_fs_cdr* entre las *N-1 y N*. Una vez encontrado el resultado en dicha tabla recogemos la información de relevancia (rosa) y la guardamos en forma de filas en la tabla ***Calls*** (azul), obteniendo el resultado de la llamada por parte de *FreeSwitch* y con la información disponible a la espera de que el terminal Android la pida.

3.5 Aplicación Android

3.5.1 Investigación Previa

La necesidad de extraer información de terminales móviles abarcando el mayor número de usuarios posibles nos lleva al desarrollo de una aplicación Android.

El hecho de que la funcionalidad principal de la aplicación sea extraer información de llamadas, nos lleva a plantearnos la utilización del fichero *Log* de Android en vez de tener la aplicación siempre escuchando a la espera de llamadas entrantes.

Todos los móviles con sistema operativo Android tienen un fichero *Log* en el que se guarda información (día, hora, fecha, duración, número llamado…) referida a las últimas 500 llamadas (entrantes o salientes) del terminal. Como particularidad del sistema, en caso de que el móvil esté mal sincronizado con el servidor NTP (*Network Time Protocol*), las horas y fechas estarán mal guardadas en dicho fichero. Un hecho a tener en cuenta en la búsqueda de información que realizara nuestra aplicación.

Para el desarrollo de la aplicación se ha utilizado Android-Studio [2] aprovechando las funcionalidades implícitas que nos otorga dicha aplicación, sobre todo en el desarrollo de *Activities* (pantallas) mediante código XML.

3.5.2 Requisitos

3.5.2.1 Requisitos funcionales.

- El sistema debe permitir identificar a los usuarios
- El sistema debe permitir registrar a los usuarios
- El sistema debe permitir enviar información
- El sistema debe permitir visualizar la información

3.5.2.2 Restricciones

- La aplicación debe estar realizada en Android(Java)

3.5.2.3 Diccionario de datos:

- Información: datos de las llamadas recibidas por el terminal
- Identificar: el sistema identifica a los usuarios mediante usuario y contraseña

3.5.3 Base de datos

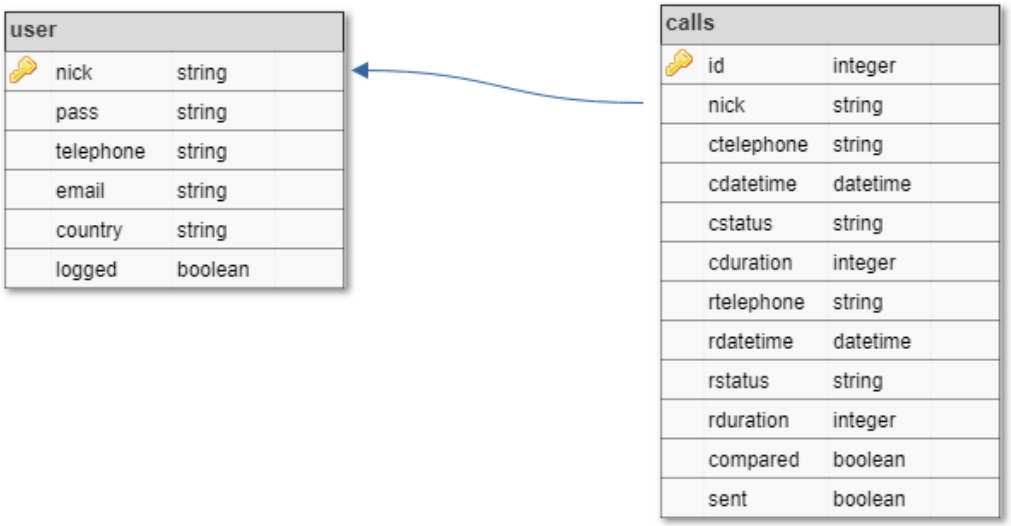


Figura 11: Diagrama relacional de las tablas de la base de datos de la aplicación móvil

La necesidad de representar en el terminal información de las llamadas que ha recibido por parte del generador así como de enviar la información de sus registros únicamente cuando este tenga conexión a internet, nos obliga a la creación de una base de datos local en la que guardemos información tanto de la llamada como del proceso de comparación de la misma. Para ello, se ha utilizado *SQL Lite* como base de datos de la aplicación Android.

La tabla *user* tiene como clave primaria el *Nick* del usuario que se ha registrado en la aplicación. Esta tabla contiene toda la información necesaria de los usuarios que se han identificado en la aplicación. A continuación se detallan los parámetros de la tabla:

- **Nick:** clave primaria de la tabla y *Nick* del usuario registrado.
- **Country:** País del usuario.
- **Email:** correo del usuario.
- **Telephone:** teléfono del usuario.
- **Pass:** contraseña del usuario.
- **Logged:** indica si el usuario guardado se ha *loggeado.*

La tabla *Calls* contiene de las llamadas que han sido analizadas o que se van analizar. A continuación se detallan los parámetros de la tabla:

- **Id:** Clave primaria de la tabla.
- **Cdatetime:** día y hora de la llamada realizada por parte de *Freeswitch.*
- **Cstatus:** información del estado de la llamada por parte de *Freeswitch* (*offhook, missed, rejected, notconnected*).
- **Ctelephone:** número con el que *Freeswitch* ha realizado la llamada.
- **Cduration:** información de la duración de la llamada por parte de *Freeswitch*.
- **Rdatetime:** día y hora de la llamada recibida por parte del terminal Android.

- **Rstatus:** información del estado de la llamada por parte del terminal Android (*offhook, missed, rejected, notfound*).
- **Rtelephone:** número que recibe el terminal Android.
- **Rduration:** información de la duración de la llamada por parte del terminal android.
- **Called_numer:** número al que se ha realizado la llamada.
- **Compared:** booleano que indica si la llamada ha sido contrastada por el terminal Android.
- **Sent:** booleano que indica si la fila de la tabla ha sido enviada al servidor.

3.5.4 Diseño

La aplicación se compone de cuatro *Activities* o pantallas. Cada una de las *Activities* se compone de una clase java en la que se define la funcionalidad de la pantalla, y un fichero XML en el que se define el aspecto de la misma.

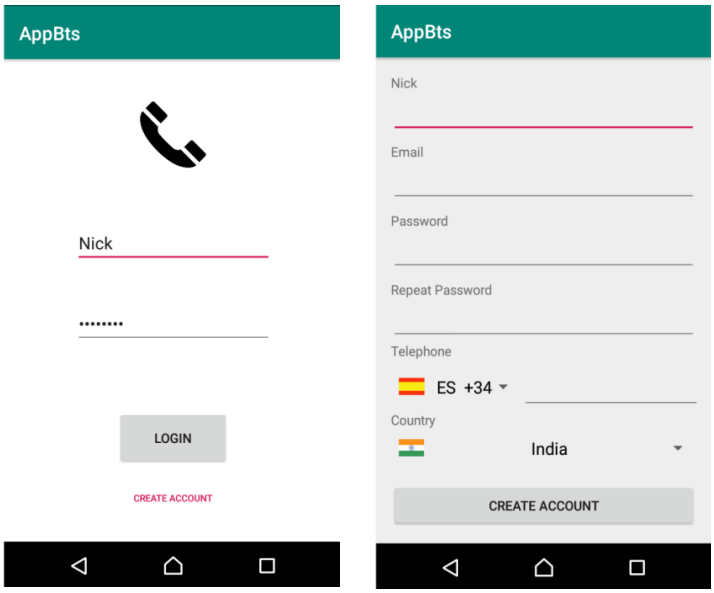


Figura 12: ***Activities Login y Create Account***

- **Login:** cuando iniciamos se muestra un *login* como el que aparece en pantalla. Si el usuario no ha sido registrado previamente deberá seleccionar la opción *Create account*. Si el usuario ya ha sido *logeado* previamente, cuando el usuario inicie la aplicación no pasará por la pantalla *Login* y saltará a la actividad *Navigation*.
- **Create Account:** se ha creado un menú sencillo para que el usuario introduzca los parámetros mínimos indispensables que necesitamos para que pueda ser registrado. Si el usuario es registrado correctamente la aplicación saltará a la pantalla *Login*.

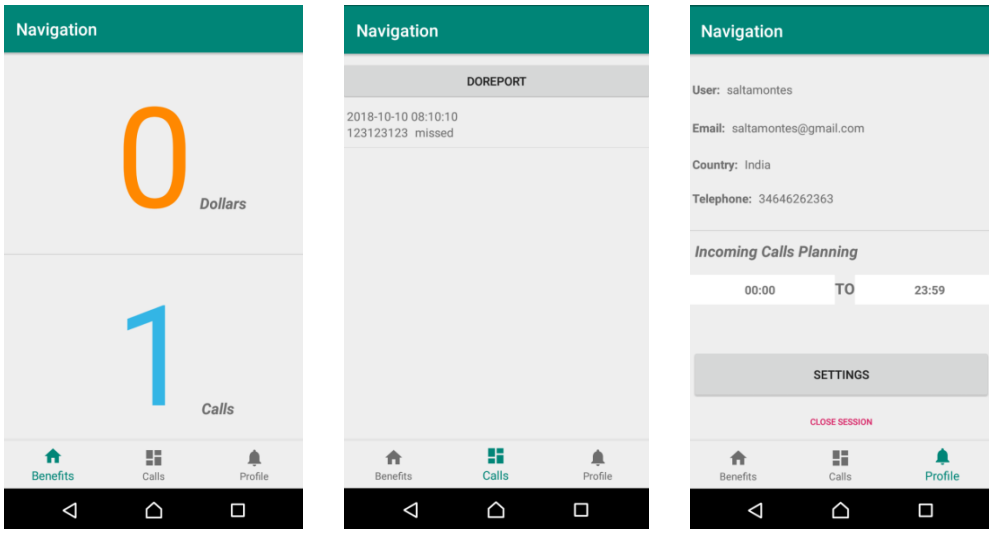


Figura 13: ***Activity Navigation***

Como podemos observar la actividad *Navigation* es una pantalla que contiene información que el usuario puede visualizar. Sobre la misma pantalla se ha supuesto un menú con tres contenidos diferenciados:

- **Benefits:** contiene información sobre la cantidad de llamadas que ha recibido el usuario así como del beneficio supuesto que ha obtenido por todas ellas (no implementado en la versión de este Trabajo Fin de Grado).
- **Calls:** tiene una visualización de una lista *Scrollable* de las llamadas que ha recibido por parte de *FreeSwitch*. El botón ***doreport*** esta implementado para intercambiar información con el servidor cuando este se pulse.
- **Profile:** contiene una visualización de los ajustes establecidos en el usuario. Si el usuario quiere cambiar los ajustes de recepción de llamadas deberá ir a *Settings*. En caso de que el usuario quiera cerrar sesión deberá pulsar *Close session*, volviendo a la pantalla de *Login*.

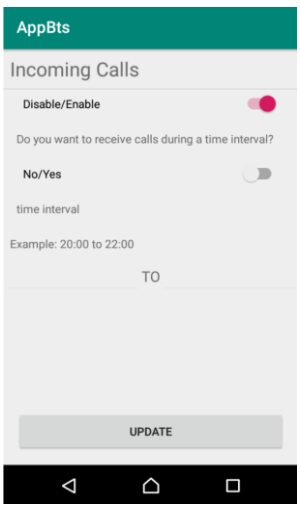


Figura 14: ***Activity Settings***

Cuando pulsemos *Settings* se abrirá una nueva actividad en la que podemos deshabilitar que nos realicen más llamadas o incluso poner un intervalo de horas en el que permitimos que nos realicen las llamadas. Esta actividad no es funcional en la versión de la aplicación de este Trabajo Fin de Grado.

3.5.5 Funcionamiento

3.5.5.1 Máquina de estados

La función principal de la aplicación radica en enviar un reporte de las llamadas que le ha realizado nuestro generador de llamadas. Para ello, se ha desarrollado una lógica basada en una máquina de estados a raíz de la pulsación del botón ***doreport*** de la segunda pestaña de la actividad *Navigation*.

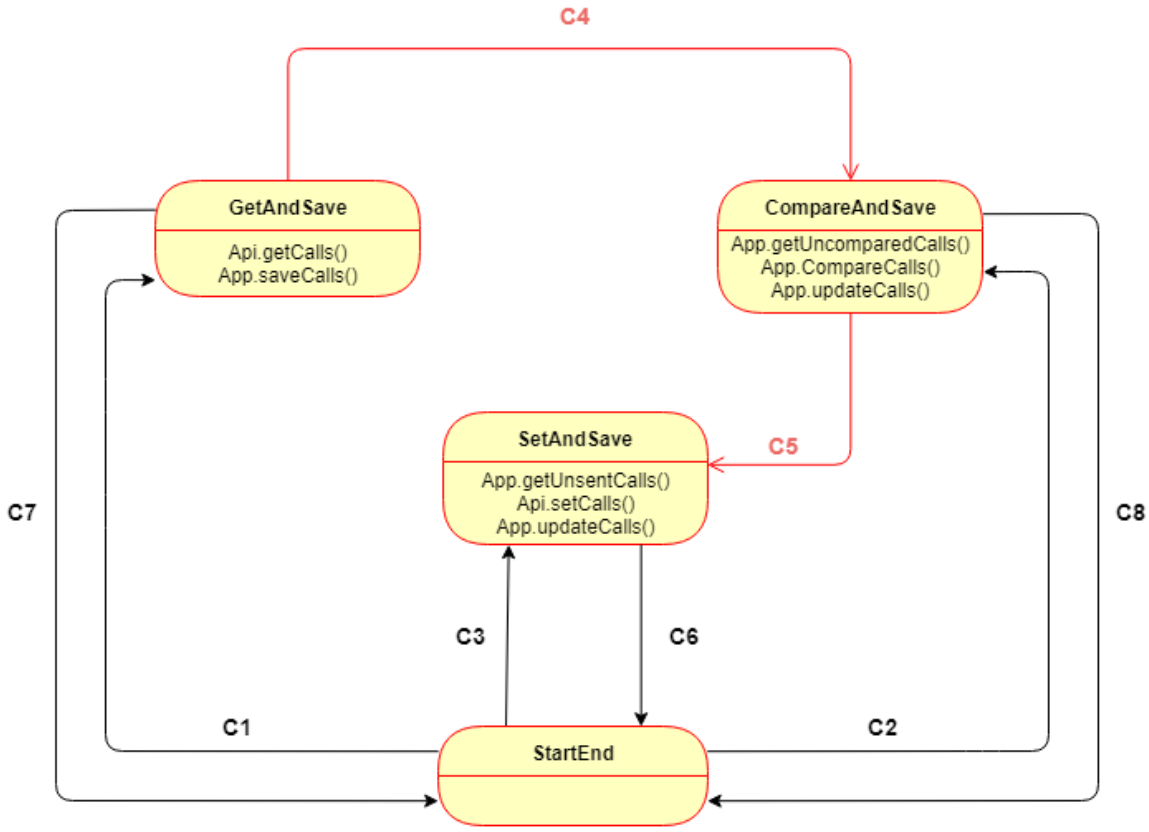


Figura 15: Máquina de estados

Las condiciones para cambiar de estado aparecen en el diagrama como "Ci", con "i" como índice de la condición.

- C1: Se ha pulsado *doreport*. No Hay llamadas sin comparar ni llamadas sin enviar en la base de datos local. Además, hay conexión a internet.
- C2: Se ha pulsado *doreport*. Hay llamadas sin comparar pero no hay llamadas sin enviar en la base de datos local. Además, hay conexión a internet.

- C3: Se ha pulsado *doreport*. No hay llamadas sin comparar pero sí que hay llamadas sin enviar en la base de datos local. Además, hay conexión a internet.
- C4: Se han obtenido llamadas por parte del servidor y tenemos conexión a internet.
- C5: Se han comparado y guardado las llamadas en la base de datos local. Además, hay conexión a internet.
- C6: Se han enviado la información de las llamadas al servidor y se ha actualizado la base de datos
- C7: Se ha guardado la información enviada por el servidor pero no tenemos conexión.
- C8: Se ha guardado la comparada por la aplicación pero no tenemos conexión para enviarla al servidor.

En el caso de que el terminal móvil se quede sin conexión de internet la máquina de estados volverá al estado *StartEnd*. Cuando el móvil tenga cobertura con internet se ejecutara *doReport* automáticamente y se volverá al estado que nos habíamos quedado anteriormente, evaluando las condiciones *C1*, *C2* y *C3*.

En el anexo "Ejemplo de funcionamiento: Aplicación " se muestra un ejemplo de funcionamiento de la máquina de estados en la aplicación móvil.

3.5.5.2 Algoritmo de búsqueda de llamadas

El procedimiento para encontrar las llamadas en el terminal Android parte de la base de que está correctamente sincronizado con un servidor NTP. Para ello, se ha establecido un cliente NTP [7] para ajustar errores del reloj del terminal Android y proceder la búsqueda en el *log* con mayor precisión.

La necesaria sincronización se debe a que como no sabemos si se ha mantenido el número llamante cuando hemos recibido la llamada, debemos buscar los datos en el fichero *log* del terminal por la fecha y hora.

El procedimiento seguido en la búsqueda de las llamadas consiste en recoger los datos de hora y fecha que de la llamada que nos ha hecho el servidor (nos los ha enviado al pulsar *doReport)* y recorrer el fichero *log* en busca de la llamada cuya fecha y hora coincidan, con un margen de error de 30 segundos. Una vez encontrada, guardamos el número de teléfono recibido, la hora y fecha exacta, la duración y el estado (perdida, contestada, rechazada o no encontrada).

Dicho procedimiento se realiza para cada una de las llamadas que nos ha hecho *FreeSwitch*. Cabe la posibilidad de que la llamada no se encuentre, bien porque el terminal no tenía cobertura o bien porque el proveedor no ha sabido conectar la llamada. En tal caso se guardara en ***Status*** *notfound*.

Capítulo 4: Implantación en varias ubicaciones

El desarrollo realizado nos proporciona un sistema flexible y portable que permite la realización de pruebas reales sin la necesidad de instalar las aplicaciones en una localización fija.

La aplicación Android se ha instalado sobre terminales Android con versión igual o superior a la *5.1*. El servidor ha sido desarrollado en *Java 8* por lo que cualquier máquina que tenga instalada dicha versión de java podrá ejecutarlo. El anexo "Guía de instalación: Servidor" detalla la configuración para la puesta en marcha del servidor sobre cualquier máquina Linux.

La portabilidad que nos otorga este desarrollo nos ha permitido la implantación del sistema sobre diferentes escenarios con el objetivo de la realización de pruebas de la manera más flexible posible.

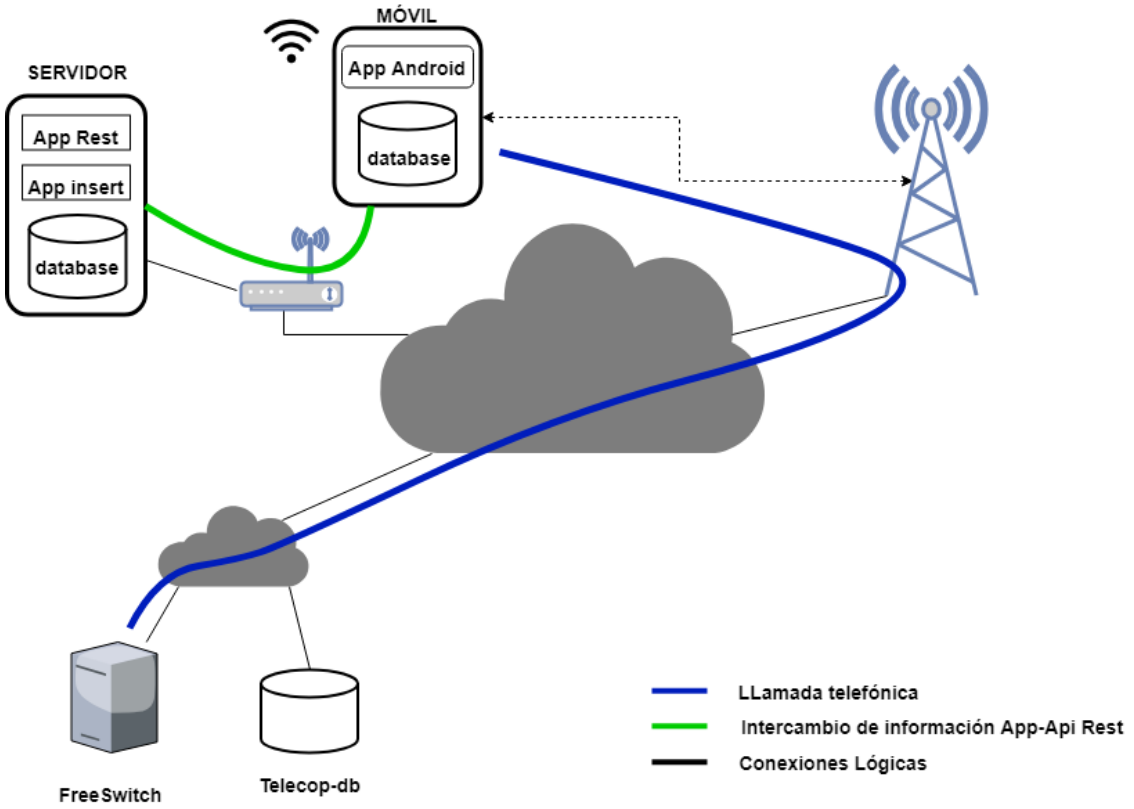


Figura 16: Escenario de pruebas local

En primer lugar, durante el desarrollo del sistema se implemento el servidor sobre un portátil *HP* con un procesador *Intel Core i3* y *4 Gigas* de *RAM*. En este caso, el servidor localizado sobre una red doméstica y con una IP privada únicamente podía intercambiar mensajes con la aplicación Android instalada en mi móvil personal a través de la red *Wifi* (en verde).

Con el desarrollo completado, se decidió instalar el servidor sobre una *Raspberry pi 3* con el único objetivo de poder realizar pruebas durante cualquiera de las 24 horas del día. El servidor funcionaba perfectamente sobre un dispositivo con mucha menos

potencia de hardware pero seguía existiendo la importante limitación de movilidad que se nos planteaba en el caso anterior. Los móviles debían estar sobre la red local para comunicarse con el servidor.

Esta pequeña limitación nos planteó la idea de instalar el servidor sobre una máquina desde la que todos los móviles con la aplicación pudieran conectarse.

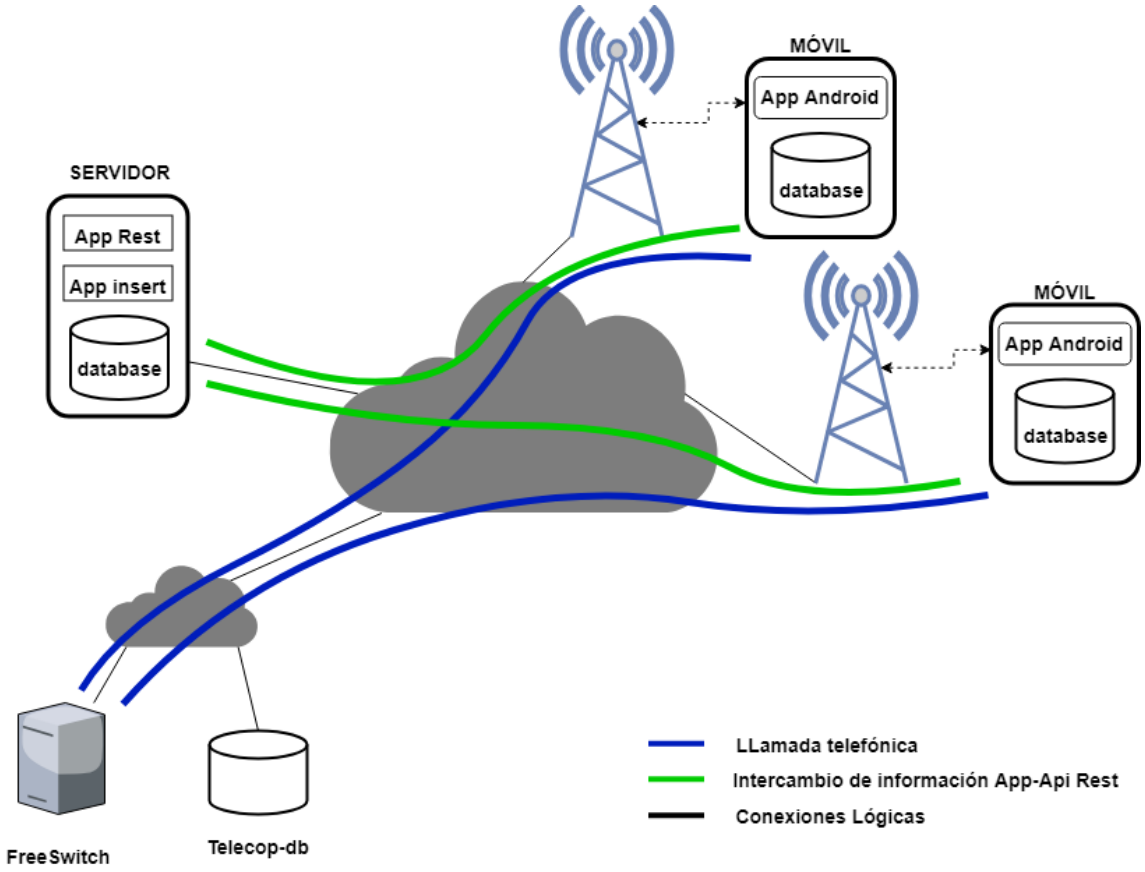


Figura 17: Escenario de pruebas final

En el último escenario se instaló el servidor sobre una máquina virtual *Ubuntu 18.04* ubicada en Teruel, propiedad de la Universidad. Dicha máquina está conectada a la red *Iris* y tiene asignada una IP pública, por tanto, es accesible desde cualquier parte del mundo. Ahora sí, la aplicación se instaló sobre varios terminales de sujetos que se prestaron a la realización de las pruebas, sin ninguna limitación sobre su ubicación física y con el único requisito de que el terminal móvil tuviera conexión a internet.

Capítulo 5: Pruebas

5.1 Pruebas durante el desarrollo

Desde el inicio del desarrollo se han realizado pruebas aisladas de cada aplicación del sistema utilizando diversas herramientas cuyo objetivo era simular el funcionamiento de aplicaciones que aún no habían sido desarrolladas, o al menos no completamente.

Durante el desarrollo de la aplicación *Rest* se realizaron comprobaciones de funcionamiento mediante la herramienta *Postman*, simulando así la realización de peticiones *http* que realizaría la aplicación móvil cuando estuviera totalmente desarrollada. Si la petición exigía guardar información en la base de datos se comprobaba si se habían actualizado las tablas mediante *Pgadmin* o a través de la consola.

Una vez se consiguió que la *aplicación Rest* funcionase correctamente se realizaron pruebas sustituyendo la herramienta *Postman* por la aplicación Android desarrollada. De esta forma, sabíamos que si se intercambiaban mal los paquetes sería debido a que la aplicación Android no estaba realizando correctamente las peticiones. Así pues, para verificar que la aplicación enviaba los paquetes correctamente se utilizó *Wireshark* con el objetivo de analizar las peticiones que llegaban al servidor.

Con la aplicación ya desarrollada se comprobó su funcionamiento sobre dos versiones de Android diferentes, la 5.1 y la 9.0, realizando pruebas y adaptando el código a los requisitos de las últimas versiones de Android, mucho más estrictas en cuanto a permisos.

Para la programación de las llamadas se comprobó mediante *Pgadmin* que las llamadas leídas del fichero se introducían en forma de filas tanto en la tabla *probecall* de la base de datos de *Telecop* como en la tabla *privateprobecall* de nuestra base de datos, y que se recogían los resultados de llamadas (tabla *bts_fs_cdr* de *Telecop*) insertándose en nuestra tabla *calls*.

Comprobado el funcionamiento por separado de cada aplicación del sistema, se procedió a la realización de pruebas globales con todas las aplicaciones en funcionamiento.

5.2 Pruebas de análisis

Para la realización de las pruebas se trabajó sobre el escenario planteado en "Figura 17: Escenario de pruebas final" instalando la aplicación móvil desarrollada sobre tres terminales móviles de diferentes compañías de telefonía.

En el primer tipo de pruebas se procedió a la realización de llamadas utilizando un número de la empresa como llamante y los números de nuestros tres terminales como números llamados. Las llamadas se encaminarían a través de diferentes proveedores.

A continuación, se muestra una tabla que representa numeradamente, de acuerdo a la nomenclatura explicada en el anexo "Nomenclatura", la lógica de realización de las llamadas.

Id	Llamante	proveedor	Llamado
1	FLX0	A	TRA1
2	FLX0	B	TRA1
3	FLX0	C	TRA1
4	FLX0	D	TRA1
5	FLX0	E	TRA1
6	FLX0	A	TRB2
7	FLX0	B	TRB2
8	FLX0	C	TRB2
9	FLX0	D	TRB2
10	FLX0	E	TRB2
11	FLX0	A	TRC3
12	FLX0	B	TRC3
13	FLX0	C	TRC3
14	FLX0	D	TRC3
15	FLX0	E	TRC3

Tabla 1: Llamadas realizadas en la primera prueba

Este ciclo de llamadas se repitió un mínimo de tres veces con el objetivo de que por proveedor algunas llamadas fueran descolgadas, otras rechazadas y el resto dejarlas como llamadas perdidas.

En el caso de las llamadas que los sujetos de pruebas no contestaron (llamadas perdidas) se observó que uno de los proveedores por los que realizamos las llamadas no tuvo un comportamiento esperado:

Called _number	Ctelephone	Cstatus	Rtelephone	Rstatus	Resource _group(provider)	Relase _cause
1	0	MISSED	-	MISSED	D	19
2	0	MISSED	-	MISSED	D	19
3	0	MISSED	-	MISSED	D	19
1	0	MISSED	99	MISSED	D	19

Tabla 2: Tabla *Calls* con llamadas perdidas

Los resultados obtenidos revelan que al realizar la llamada por el proveedor *D*, los terminales recibían las llamadas con un número privado o con un número diferente al número llamante. El resto de los proveedores escogidos se comportaron de la manera esperada.

Las llamadas que fueron rechazadas por los terminales de pruebas, además del continuo cambio de número llamante al lanzar la llamada por el proveedor *D*, se obtuvieron otros resultados complementarios:

Called _number	Ctelephone	Cstatus	Cduration	Rtelephone	Rstatus	Rduration	Resource _group	Release _cause
3	0	OFFHOOK	13	0	REJECTED	0	A	16
3	0	OFFHOOK	10	0	REJECTED	0	A	16
3	0	OFFHOOK	8	0	REJECTED	0	B	16

Tabla 3: Tabla *Calls* con llamadas rechazadas

Todas las llamadas que se hicieron al terminal *TRC3* y este rechazó, otorgaron al llamante una duración de 13, 10 y 8 segundos, cuando en realidad la llamada había sido rechazada y por tanto de duración 0 segundos. La causa de liberación de la llamada (16), nos confirma que la llamada había sido descolgada por lo que podemos asegurar que la operadora descolgaba las llamadas que su cliente rechaza.

En el caso en el que las llamadas son contestadas, se produjeron tanto cambio de números llamantes como variaciones importantes en las duraciones de la llamada:

Called _number	Ctelephone	Cstatus	Cduration	Rtelephone	Rstatus	Rduration	Resource _group	Release _cause
1	0	OFFHOOK	10	-	OFFHOOK	6	D	16
1	0	OFFHOOK	7	0	OFFHOOK	3	E	16
2	0	OFFHOOK	13	0	OFFHOOK	7	E	16
3	0	OFFHOOK	12	-	OFFHOOK	9	D	16

Tabla 4: Tabla *Calls* con llamadas contestadas

Como podemos observar, las llamadas realizadas a través de los proveedores D y E eran dadas en muchas ocasiones a tener comportamientos anómalos, ya que cambiaba el número llamante o la duración real de la llamada, estableciéndose con mayor duración de lo que en realidad era.

Finalmente, se produjo la situación en la que el proveedor B no fue capaz de conectar ninguna de las llamadas que le enviamos:

Called _number	Ctelephone	Cstatus	Rtelephone	Rstatus	Resource _group	Release _cause
1	0	NOTCONNECTED	-	NOTFOUND	C	34
2	0	NOTCONNECTED	-	NOTFOUND	C	34
3	0	NOTCONNECTED	-	NOTFOUND	C	34

Tabla 5: Tabla *Calls* con llamadas no conectadas

La causa de liberación 34 nos indica que no hay circuito o canal disponible para manejar la llamada, por lo que no pudo conectarla.

En el segundo tipo de pruebas se procedió a la realización de llamadas alternando como número llamante los números de teléfono de los colaboradores de la prueba, y los números de nuestros tres terminales como números llamados. Las llamadas se encaminarían a través de diferentes proveedores.

Id	Llamante	proveedor	Llamado
1	FLA4	A-B-C	TRA1
2	FLB2	A-B-C	TRA1
3	FLC3	A-B-C	TRA1
4	FLA1	A-B-C	TRB2
5	FLB5	A-B-C	TRB2
6	FLC3	A-B-C	TRB2
7	FLA1	A-B-C	TRC3
8	FLB2	A-B-C	TRC3
9	FLC6	A-B-C	TRC3

Tabla 6: Llamadas realizadas en la segunda prueba

Los números representados en azul son números de teléfono que colaboradores nos prestaron para realizar esta prueba, cada uno de ellos perteneciente a cada una de las tres compañías sobre las que se iba a realizar el experimento. Los números representados en verde son los números de los terminales que tienen la aplicación instalada, para poder recabar los datos de la prueba.

El ciclo de nueve llamadas se repitió tres veces, una por cada proveedor, con el objetivo de verificar el comportamiento de todos los proveedores.

A continuación, se muestra únicamente el resultado al realizar las llamadas por el proveedor A, ya que el proveedor C no conectó ninguna, y el resto de proveedores tuvieron el mismo comportamiento que el proveedor A:

Called_number	Ctelephone	Rtelephone	Resource_group(provider)
1	4	4	A
1	2	2	A
1	3	3	A
2	1	1	A
2	5	5	A
2	3	3	A
3	1	1	A
3	2	2	A
3	6	6	A

Tabla 7: Tabla *Calls* tras realizar las llamadas de la segunda prueba

Tal y como podemos observar en la tabla, todas las llamadas realizadas con un número llamante suplantado llegaron a su destinatario correctamente, sin ningún tipo de filtro sobre números de teléfono por parte de ninguno de los proveedores.

Capítulo 6: Conclusiones y trabajos futuros

6.1 Conclusiones

En este Trabajo Fin de Grado se ha diseñado e implementado un sistema para la evaluación de proveedores de telefonía IP con el objetivo de detectar anomalías en el comportamiento de las llamadas.

Se ha analizado el comportamiento de la herramienta *Telecop* con el objetivo de aprovechar el funcionamiento del generador de llamadas del que hace uso (*FreeSwitch*), adquiriendo conocimientos sobre su funcionamiento para la posterior integración con el sistema que se ha desarrollado en este proyecto.

Se han desarrollado:

- Una aplicación de tipo *Rest* que intercambia con los usuarios de la aplicación móvil información de registro, identificación y llamadas, y la guarda en la base de datos del servidor.
- Una aplicación que permite tanto programar la realización de llamadas como recoger el resultado de las mismas (duración, hora de conexión, estado…) cuando estas se realicen.
- Una aplicación Android que recoge y envía datos al servidor de las llamadas que le ha realizado *FreeSwitch*.

Se ha instalado la aplicación móvil en terminales Android de diferentes compañías y se han instalado las aplicaciones del Servidor en diferentes equipos, comprobando así la flexibilidad que otorga el tipo de desarrollo realizado.

Se ha configurado una VPN para la conexión segura entre el servidor y la base de datos de *Telecop* (ubicada en la empresa), permitiendo a la *aplicación de inserción de llamadas* actuar de manera segura.

Durante el desarrollo de este Trabajo Fin de Grado se han adquirido conocimientos sobre las herramientas *Spring*, *Android Studio* y *PostgreSql*. Además, se han ampliado los conocimientos en administración de sistemas.

No solo se han cumplido con los objetivos propuestos sino que se han llevado más allá, proporcionando un sistema funcional, flexible y escalable, con el que empresas como BTS podrán evaluar a proveedores con la consiguiente posible mejora en el servicio de llamadas, ya que dejaran de enviar tráfico por redes con comportamientos fraudulentos para pasar a enviarlo por redes seguras que garanticen cierta calidad, aumentando los beneficios de la empresa y reduciendo el gasto de los clientes.

6.2 Líneas Futuras

Una vez desarrollada la base del sistema, las líneas futuras de este proyecto van asociadas a una mejora en cuanto a la inclusión de nuevas funcionalidades así como de un análisis profundo de los comportamientos que toman las llamadas al ser enviadas por diferentes proveedores, con el objetivo de tener un sistema que la empresa pueda utilizar tanto a nivel local como comercial.

De cara a futuras actualizaciones, sin tener en cuenta la lógica comercial que llevara el sistema en el futuro, hay varios aspectos a mejorar o al menos a contemplar, ya que pueden suponer un tiempo y esfuerzo considerable.

La identificación de la aplicación en vez de hacerla de forma implícita en la aplicación se podría derivar a un tercero como *OpenId*, despreocupándonos del registro e identificación de usuarios por parte de la aplicación y centrándonos en la funcionalidad. *OpenId* nos proporciona una pantalla de *Login* que podemos incluir en nuestra aplicación, si el usuario esta registrado la *Api* de *OpenId* nos devuelve un *token* verificando su registro. Además, permite al usuario identificarse con las cuentas de *Google*, *Facebook* y *Twitter*.

Las comunicaciones del terminal Android con el servidor actualmente se realizan sobre el protocolo *http*. Será crucial la securización de dicha comunicación pasando a la utilización de *https* en una futura actualización de la aplicación.

La implementación funcional de la actividad *Settings* para que los usuarios puedan decidir las horas que tienen disponibles para que les realicen llamadas. Para ello, deberíamos añadir la funcionalidad también en la *aplicación Rest* para intercambiar información sobre la disponibilidad del usuario.

Teniendo en cuenta una posible salida comercial por parte de la empresa, como motivación a que usuarios ajenos a la empresa se instalen la aplicación se otorgarían bonificaciones a través de un sistema de pagos a cambio de recibir llamadas, con el objetivo de aumentar la cantidad de usuarios a los que podemos llamar. Si llamando al usuario se detectase una situación fraudulenta por parte de algún proveedor, entonces se le otorgaría una bonificación que aparecería en su pestaña *benefits*.

Debido a que el sistema desarrollado alberga sus resultados en una base de datos, se podría realizar una aplicación web para la visualización los mismos, así como estadísticas de las llamadas, usuarios, etc. Se podría incluir la funcionalidad de programación de llamadas desde la web, facilitando la utilización del sistema al usuario final.

A un nivel de análisis, en este trabajo fin de Grado se han hecho pruebas realizando un conjunto de llamadas limitado y sobre unos pocos proveedores. Se podría realizar un estudio de comportamientos de un conjunto más amplio de proveedores, analizando el comportamiento de las llamadas frente a diferentes situaciones que se pueden plantear.

Tal y como se puede observar, el hecho de haber desarrollado un sistema escalable y flexible nos otorga múltiples mejoras y futuros posibles que sin duda se llevarán a cabo.

Referencias

[1] Documentación de *Spring*:

https://spring.io/guides

https://www.baeldung.com/rest-with-spring-series

[2] Documentación *Android*:

https://developer.android.com/studio/intro

[3] Documentación para *Postgres*:

https://www.postgresql.org/docs/manuals/

[4] Documentación códigos *Release Cause* de *FreeSwitch*:

https://freeswitch.org/confluence/display/FREESWITCH/Hangup+Cause+Code+Table

[5] Documentación paquete *Java* :

https://www.java.com/es/download/faq/whatis_java.xml

[6] Documentación funcionamiento *NTP*:

https://es.wikipedia.org/wiki/Network_Time_Protocol

[7] Librería cliente *NTP* para *Android*:

https://github.com/instacart/truetime-android

Anexos

1.1 Red de la empresa

La red de la empresa consta de la estructura que se muestra a continuación:

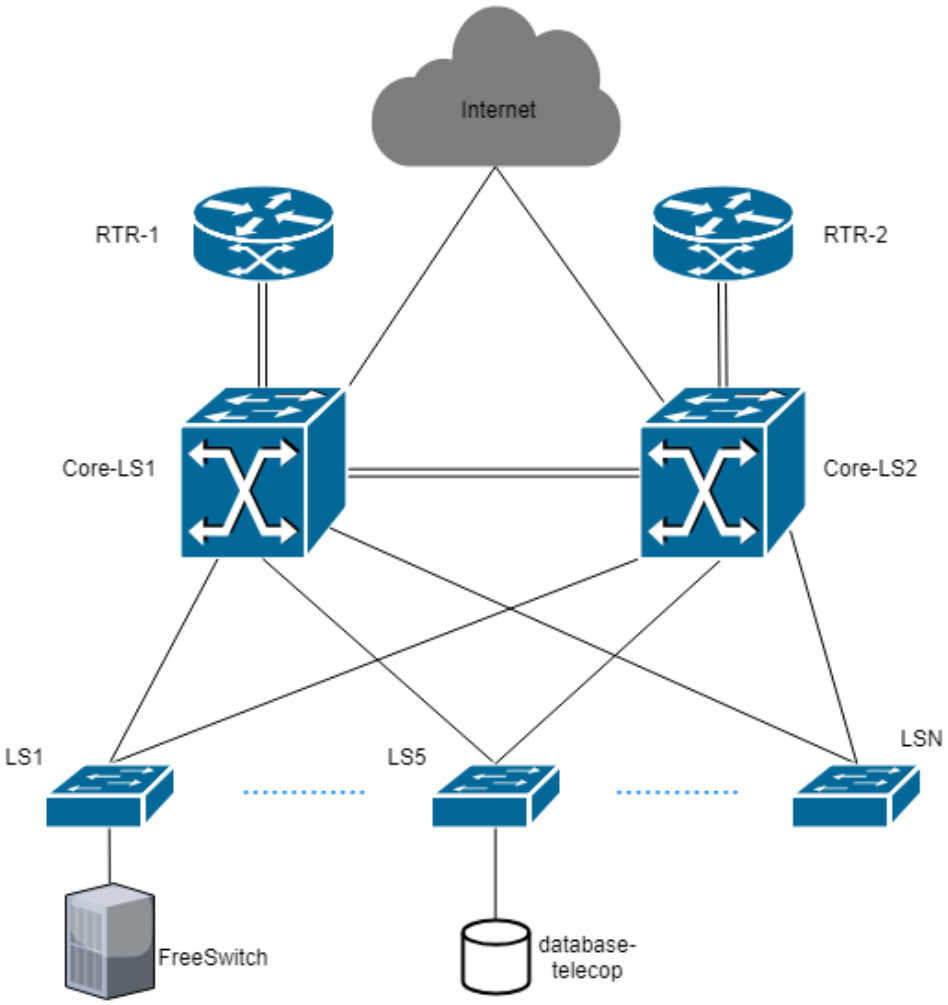


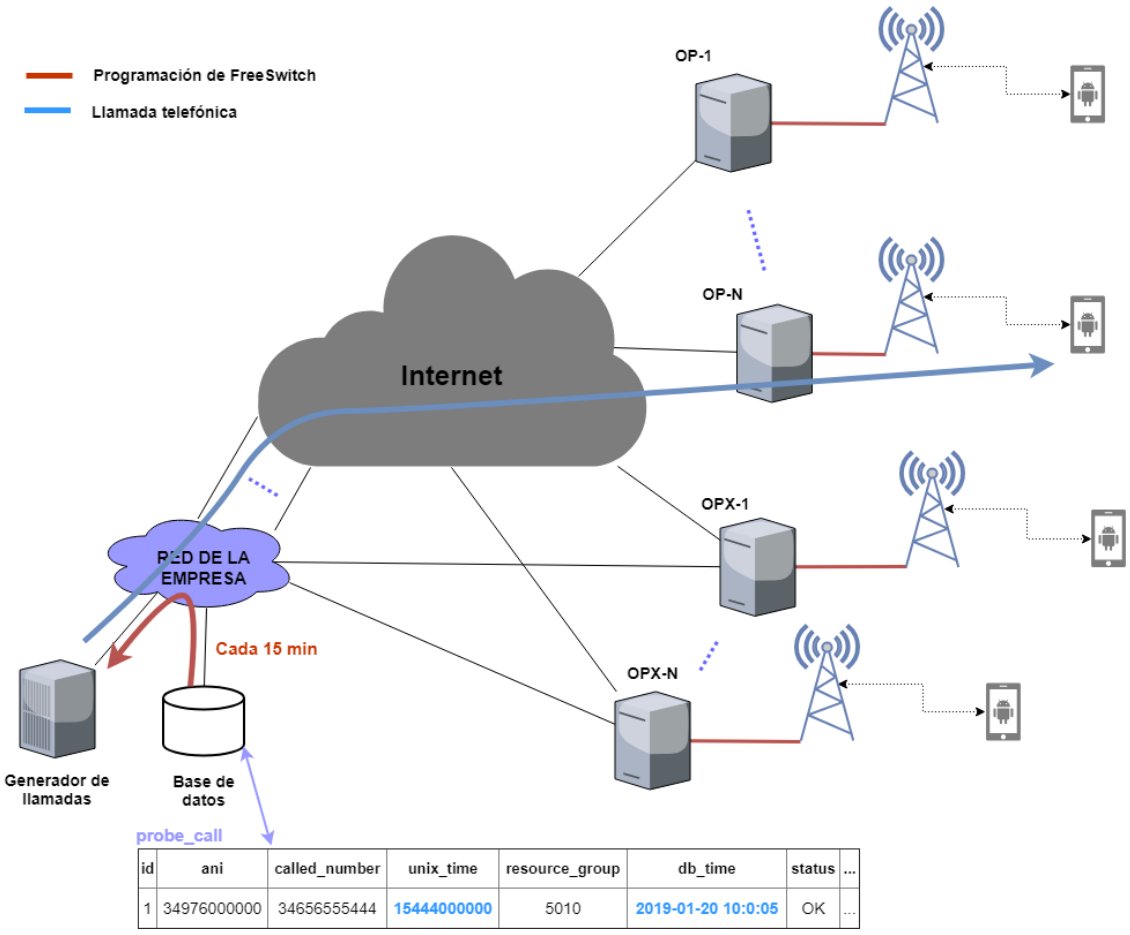
Figura 18: Red de la empresa

Los *Core-LS1* (*Lan Switches*) vy *Core-LS2* se encargan de conmutar todos los paquetes necesarios para que salgan hacia internet o hacia redes internas de la empresa. Los *Lan Switches* (LS) tienen conexiones a los equipos de la red en la que están así como a los dos *Core-LS* a modo de redundancia. Si un *Core-LS* falla siempre existirá el otro. Ambos *Core-LS* están conectados entre sí a modo de previsión de fallos de alguno de los *Router*.

La conexión que establecen los Core-LS con internet es a través de diferentes proveedores, en previsión de que alguno de ellos falle.

1.2 Ejemplo de funcionamiento: *Telecop*

El proceso de realización de llamadas parte de la escritura en la tabla *probe_call* de la base de datos de *Telecop*. Rellenando la tabla definida en el *estudio previo* con los parámetros necesarios.



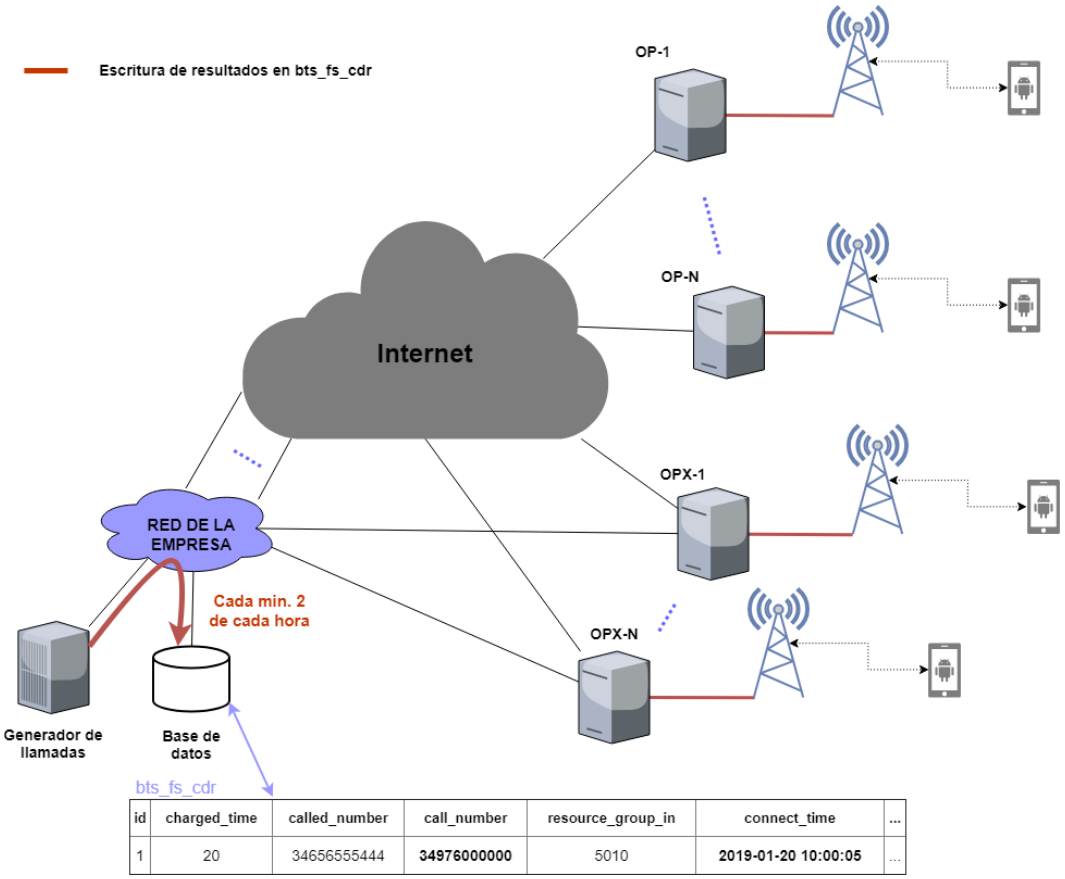
id	ani	called_number	unix_time	resource_group	db_time	status	...
1	34976000000	34656555444	15444000000	5010	2019-01-20 10:0:05	OK	...

Figura 19: Lectura base de datos y realización de llamada

En este caso llamaremos al número *34656555444* utilizando como número llamante el *34976000000*. Para ello, rellenaremos la tabla completando todos los parámetros de la misma, y sobre todo rellenando la fecha y la hora de manera correcta.

Un Script lee la tabla cada 15 minutos y recoge las filas cuya columna *Status* sea igual a *scheduled*. Si todas las columnas de la fila están rellenadas correctamente pondrá la columna Status a *ok* y programará la llamada en el *FreeSwitch*.

Cuando venza la hora y fecha indicada, *FreeSwitch* lanzará la llamada al destinatario móvil que hemos indicado y a través del proveedor que hemos establecido.



id	charged_time	called_number	call_number	resource_group_in	connect_time	...
1	20	34656555444	34976000000	5010	2019-01-20 10:00:05	...

Figura 20: Escritura en base de datos el resultado del fichero log

Con la llamada ya realizada existe otro script que recoge los resultados del fichero log (generado por *FreeSwitch*) cada minuto 2 de cada hora y los pasa a la tabla *bts_fs_cdr*. En este caso, a las 11:02 aparecerá el resultado de la llamada en dicha tabla.

Como podemos observar la columna *charged_time* nos indica que la llamada ha durado 20 segundos, y la columna *connect_time* nos indica la hora en la que la llamada ha sido conectada.

1.3 Ejemplo de funcionamiento: Aplicación Android

La mejor forma de entender el funcionamiento del *doReport* es mediante un ejemplo desde el punto de vista de la aplicación.

Cuando el usuario pulsa *doReport*, la aplicación va a la máquina de estados y mira la tabla *Calls* de la base de datos local. Suponiendo el caso en el que la tabla este vacía, la máquina de estados cumple la condición *C1* (ya que no hay llamadas sin comparar ni sin enviar) y pasa al estado *GetAndSave*.

En el estado ***GetAndSave*** realizamos una petición *http get* al servidor y guardamos el resultado en la base de datos local. Como podremos observar, inicialmente la tabla está vacía.

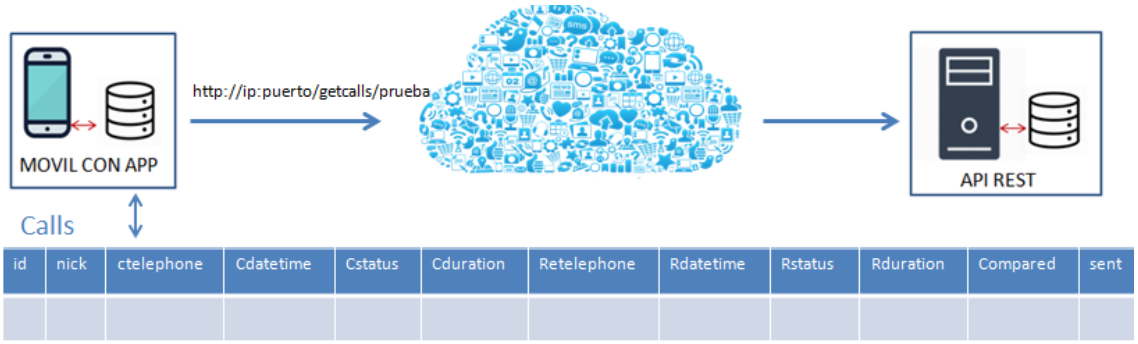


Figura 21: Petición *http get* al servidor

Recibimos la respuesta del servidor *rest* con las columnas de la tabla que nos interesan. Si nos hubieran hecho más llamadas recibiríamos tantas filas como llamadas nos hubieran realizado.

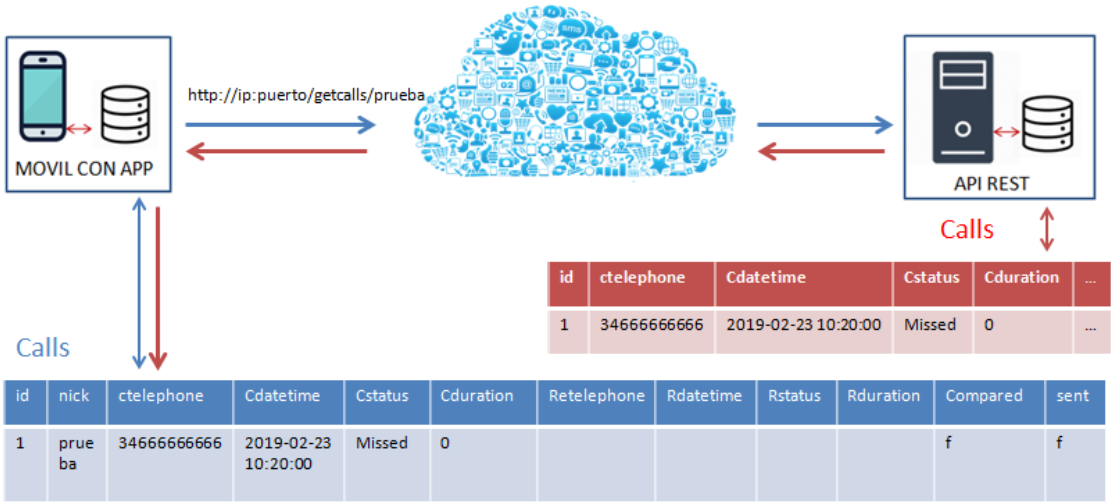


Figura 22: Respuesta del servidor y actualización de la tabla local

A continuación, el móvil tiene guardado en su base de datos local la llamada que le ha realizado el generador de llamadas con el número de teléfono, la hora y fecha en *GMT*, el estado y la duración.

Como podemos observar, la llamada se ha guardado con los parámetros *compared* y *sent* igual a *false* debido a que hemos recibido la fila pero no se ha comparado con el log del teléfono ni se ha enviado de vuelta al servidor.

Cumpliendo la condición *C4* pasamos al estado ***CompareandSave***. En dicho estado, el terminal busca en el *log* las llamadas cuya hora y fecha en *GMT* coincidan con el parámetro *Cdatetime* de la tabla de la base de datos (con un margen de error de 30 segundos). Una vez encontrada dicha llamada cogemos los parámetros de número de teléfono, hora y fecha, duración y *status*, y los pasamos a la tabla de la base de datos.

id	nick	ctelephone	Cdatetime	Cstatus	Cduration	Retelephone	Rdatetime	Rstatus	Rduration	Compared	sent
1	prueba	34666666666	2019-02-23 10:20:00	Missed	0	34666666666	2019-02-23 10:20:02	Missed	0	t	f

Tabla 8: Llamada comparada pero no enviada

Como podemos observar se ha modificado la tabla (rojo) y ahora tenemos todas las columnas completadas con el resultado de la búsqueda en el *log* del terminal. Además, la columna *compared* ha pasado a valer *true* ya que se ha realizado el proceso de comparación por parte del terminal.

En este estado y cumpliendo la condición *C5* pasaríamos al estado ***SetAndSend***. En este caso, el terminal realizará una petición *http* de tipo *post* al servidor, enviando todas las filas de la tabla en las que *compared* sea *true* y *sent* sea *false*. Completado dicho proceso se actualizará la tabla y *sent* pasará a valer *true*.

id	nick	ctelephone	Cdatetime	Cstatus	Cduration	Retelephone	Rdatetime	Rstatus	Rduration	Compared	sent
1	prueba	34666666666	2019-02-23 10:20:00	Missed	0	34666666666	2019-02-23 10:20:02	Missed	0	t	t

Tabla 9: Llamada comparada y enviada

En el caso de que por cualquiera de las condiciones se volviera al estado *StartEnd*, la próxima vez que se ejecute *doReport* volveríamos al estado que nos hubiéramos quedado, comparando o enviando las filas de la base de datos antes de volver a pedir nueva información de las llamadas.

1.4 Nomenclatura

Con el objetivo de plasmar las pruebas de la mejor manera posible y de mantener la confidencialidad tanto de los usuarios como de los proveedores se ha utilizado una nomenclatura que representa una abstracción del equipo que recibe o realiza llamadas en función de sus características:

- Tipo: *FreeSwitch*(F) o Terminal(T)
- Función: realizar llamadas(L) o recibir llamadas(R)
- Compañía: A, B, C, D, E …
- Número de teléfono: 0,1, 2, 3, 4 …

De esta manera, el conjunto de características de un equipo que realiza llamadas seria por ejemplo el siguiente:

Tipo	Función	Compañía	Nº de Teléfono
F	L	D	0

Tabla 10: Ejemplo de equipo llamante

El equipo descrito es el *FreeSwitch* con la función de llamar con el número de teléfono "0". Dicho número pertenece a la compañía "D".

A continuación, el conjunto de características de un equipo que recibe llamadas sería por ejemplo el siguiente:

Tipo	Función	Compañía	Nº de Teléfono
T	R	A	1

Tabla 11: Ejemplo de equipo llamado

El equipo es un terminal móvil cuya función es recibir llamadas. Su número de móvil es el "1", y pertenece a la compañía "A".

1.5 Códigos *Release Cause*

Para la realización de este anexo se ha seguido la norma que aporta *FreeSwitch* en su documentación [5].

La columna *release_cause* de la tabla ***calls*** contiene un número que aporta información sobre lo que ha ocurrido con la llamada por parte de *FreeSwitch*. A diferencia de la columna *status*, *release_cause* nos aporta información específica de la causa de liberación o conexión de las llamadas.

A continuación, se detalla una tabla con el significado de cada uno de los principales códigos posibles.

ITU-T Q.850 Code	SIP Equiv.	Enumeración	Causa	Descripción
16		NORMAL_CLEARING	Liberación normal de la llamada	Esta causa indica que la llamada se está liberando porque uno de los usuarios de la llamada ha solicitado que esta se libere.
17	486	USER_BUSY	Usuario ocupado	Esta causa se utiliza para indicar que el receptor de la llamada no puede aceptar la misma porque el usuario se encuentra en condición desocupado
18	408	NO_USER_RESPONSE	Usuario no responde[Q.850]	Esta causa se utiliza cuando una parte de la llamada no responde a un mensaje de establecimiento de llamada.
19	480	NO_ANSWER	No hay respuesta del usuario (usuario avisado)	Esta causa se utiliza cuando el receptor ha sido avisado pero no responde con una indicación de conexión dentro de un tiempo prescrito.

21	603	CALL_REJECTED	Llamada rechazada	Esta causa indica que el terminal no desea aceptar la llamada. La red también podría generar esta causa, lo que indica que la llamada se liberó debido a una restricción de servicio.
22	410	NUMBER_CHANGED	Número cambiado	Esta causa se da cuando el número llamado no está asignado.
25	483	EXCHANGE_ROUTING_ERROR	Circuito/Canal no disponible	Esta causa indica que no se puede llegar al destino indicado por el usuario, debido a que la central ha liberado la llamada porque ha alcanzado un límite en la ejecución del contador de saltos
34	503	NORMAL_CIRCUIT_CONGESTION	Circuito/Canal no disponible	Esta causa indica que no hay circuito/canal disponible para manejar la llamada

Tabla 12: Códigos *release cause*

1.6 Guía de instalación: Servidor

Para poder ejecutar el código del servidor será necesaria la instalación de una base de datos local con *postgresql*, una *vpn* con *OpenVpn* y de la versión 8 de *java* en caso de que no esté instalada. Además de configurar las aplicaciones mencionadas será necesario establecer en *GMT* la zona horaria de la máquina *Ubuntu* que ejecuta nuestro código.

En primer lugar instalamos el gestor de bases de datos *postgresql*:

- ***sudo apt-get install postgresql***

Una vez instalada debemos acceder al programa y crear una base de datos denominada ***bts***. Para acceder a *Postgres* debemos cambiar al usuario *postgres* que se crea por defecto. Sobre el terminal:

- ***sudo -i -u postgres***

Una vez ya establecidos como usuario *postgres* debemos ejecutar el comando ***psql*** para acceder al programa. Dentro de dicho programa y antes de ejecutar nada debemos configurar una contraseña por defecto para poder acceder remotamente. Para ello ejecutamos:

- ***\password***

y ponemos de contraseña

- ***postgres***

Al tener configurados el **usuario *postgres*** y la **contraseña *postgres***, el servidor podrá acceder remotamente a la base de datos. A continuación, ya podemos proceder a la creación de una base de datos. Para ello, dentro del programa ejecutamos:

- ***CREATE DATABASE bts;***

Una vez creada, será necesaria la creación de las tres tablas asociadas a dicha base de datos, las tablas ***users***, ***real_probe_call***, y ***calls***.

Para ello, sobre la consola postgres y con el programa ejecutado accedemos a la base de datos mediante el comando ***\c bts*** y creamos las tablas copiando el código, pegándolo sobre el terminal y ejecutando *intro*.

```
CREATE TABLE public.users
(
  nick character varying(15) NOT NULL,
  active boolean,
  country character varying(255),
  email character varying(255),
  ncalls integer,
  pass character varying(255),
  telephone character varying(255),
  CONSTRAINT users_pkey PRIMARY KEY (nick)
);

CREATE TABLE public.real_probe_call
(
  id bigserial,
  version bigint DEFAULT 1,
  ani character varying(255),
  area_code character varying(255),
  call_battery_id bigint,
  called_number character varying(255),
  destination character varying(255),
  resource_group integer,
  status character varying(255),
  unix_time bigint,
  probe_calls_idx integer,
  db_time timestamp with time zone,
  answer boolean,
  CONSTRAINT real_probe_call_pkey PRIMARY KEY (id)
);
CREATE TABLE public.calls
(
id bigserial,
cdatetime timestamp without time zone,
compared boolean,
cstatus character varying(255),
ctelephone character varying(255),
rdatetime timestamp without time zone,
rstatus character varying(255),
rtelephone character varying(255),
nick character varying(15),
cduration integer,
rduration integer,
called_number character varying(255),
resource_group integer,
release_cause integer,
CONSTRAINT calls_pkey PRIMARY KEY (id),
CONSTRAINT nick FOREIGN KEY (nick)
REFERENCES public.users (nick) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION);
```

Tenemos la base de datos creada y las tablas definidas pero aunque hemos añadido una contraseña al usuario *postgres* no lo hemos configurado como tal para permitir su acceso remoto. Para ello, salimos de *Postgres* mediante la ejecución de **\q** y con el usuario ***root*** vamos al siguiente directorio:

- ***/etc/postgresql/10/main/***

Abrimos el fichero ***postgresql.conf*** y debemos descomentar ***listen_addresses*** y poner la ip de ***localhost.*** Si el puerto esta comentado debemos poner el *5432* (***port = 5432***). Además, en la linea *timezone* sería necesario poner ***timezone = 'GMT'.*** Cambiados dichos parámetros salimos y guardamos.

Sobre el mismo directorio debemos abrir el fichero ***pg_hba.conf*** y descomentar las siguientes líneas:

 # Allow replication connections from localhost, by a user with the

 # replication privilege.

local replication all peer

host replication all 127.0.0.1/32 md5

host replication all ::1/128 md5

Una vez descomentadas dichas líneas debemos guardar el fichero y para que *potsgres* aplique los cambios debemos ejecutar el siguiente comando:

- ***service postgresql restart***

Para comprobar que todos los cambios se han efectuado debemos en primer lugar comprobar que se ha cambiado la hora de la base de datos. Para ello, accedemos a *postgresql* (como lo hemos hecho anteriormente) y ejecutamos el siguiente comando:

- ***select now();***

Dicho comando debería darnos la hora en *GMT*. Si la hora y fecha que da es la correcta, podemos proceder a comprobar si tenemos acceso remoto a la base de datos. Para ello debemos ejecutar sobre una terminal el siguiente comando:

- ***psql -h 127.0.0.1 -U postgres -d bts***

Si nos pide la contraseña, la introducimos y entramos es que todo ha funcionado correctamente y por consiguiente ya está configurada la base de datos para que el servidor la utilice.

Configurada la base de datos que utilizara el servidor, debemos configurar la hora y fecha en *GMT* de la máquina sobre la que se ejecutara la aplicación. Para ello ejecutamos sobre el terminal el siguiente comando:

- ***sudo timedatectl set-timezone UTC***

Para que la aplicación instalada pueda conectarse con la base de datos de *telecop*, debemos instalar y configurar una vpn con unas credenciales proporcionadas por BTS. Para ello, en primer lugar instalamos openvpn con el siguiente comando:

- ***sudo apt-get install openvpn easy-rsa***

Cuando se haya instalado debemos introducir los 5 archivos del *zip* en el directorio */etc/openvpn/* y cambiar la extensión del archivo ***client.ovpn*** por la de ***client.conf***. Dichos archivos han sido otorgados por el gestor de VPN, en este caso la empresa BTS. Con todo configurado, solo queda iniciar el servicio:

- ***Service openvpn start***

Con la base de datos, la *vpn*, y la hora y fecha de la máquina configuradas, solo falta asegurarnos de que la versión de java que tenemos instalada sea la Java versión 8. Para ello ejecutamos el siguiente comando:

- ***javac –version***

Si nos dice que tenemos la versión 8 de java podemos pasar a ejecutar la aplicación, sino, debemos instalar instalarla. Para ello debemos ejecutar los siguientes comandos:

- ***sudo add-apt-repository ppa:webupd8team/java***
- ***sudo apt-get update***
- ***sudo apt-get install oracle-java8-installer***

Con el paquete de java instalado solo falta asegurarnos de que utiliza Java 8 aunque tenga instalados otros paquetes. Para ello ejecutamos el siguiente comando:

- ***sudo update-alternatives --config java***

Seleccionamos la versión 8 de java y ya podremos lanzar la aplicación.

Si por algún casual los cambios no se aplican es necesario reiniciar la máquina. En caso de haber reiniciado la máquina se recomienda revisar que tenemos correctamente configurado el *servidor dns*, ya que nuestra aplicación se conecta a una base de datos remota y resuelve su IP a través del servidor de nombres.

Con todo lo anterior configurado solo queda ejecutar el fichero *jar*. El fichero *JAR* debe tener en el mismo directorio un fichero *p.txt* en el que se introducirán las llamadas. Para ejecutar la aplicación únicamente será necesario lanzar el siguiente comando:

- ***java -jar archivo.jar***

Cuando la aplicación arranque nos pedirá una la IP del servidor y el puerto en el que queremos que escuche las peticiones *http*. Introducimos los parámetros y finalmente el servidor se pondrá en funcionamiento.

1.7 Guía de usuario: Servidor

Una vez lanzado el servidor es necesario conocer como introducir llamadas para que se programen en *FreeSwitch* y se realicen a la hora y fecha que al usuario le interese.

Para ello, sobre la carpeta en la que hemos lanzado el servidor debemos crear un fichero con nombre **"p.txt".** Sobre dicho fichero introduciremos en forma de filas las llamadas que queremos realizar. A continuación, se muestra como se representa una fila en el fichero, para conocer como debería rellenarse:

- ***XXXXXXXXX,YYYYYYYY,2019-01-26 16:50:00,5010,255***

En la primera columna de la fila de debemos introducir el número llamado, es decir, el número al que queremos realizar la llamada.

En la segunda columna de la debemos introducir el número llamante, es decir, el número que va a utilizar *FreeSwitch* para realizar la llamada.

En la tercera columna debemos introducir la fecha y hora a la que queremos que se realice la llamada. No se pueden hacer llamadas para dentro de 5 minutos, se deben planificar con al menos media hora de antelación. El formato de fecha y hora debe ser el especificado en el ejemplo.

La cuarta columna es el identificador del proveedor por el que vamos a realizar la llamada. Dicho identificador es recogido de una tabla que tiene la empresa en la que asocia cada identificador a una compañía.

La quinta columna representa el código de área de un proveedor. Un proveedor tiene varios lugares por los que conectar la llamada, debemos elegir uno de entre todos los que tiene como requisito para que se realice.

Debemos introducir tantas filas con el formato establecido como llamadas queramos realizar. Cada 2 minutos la aplicación recogerá las filas y las introducirá en forma de llamadas en la base de datos de *Telecop*, programando así las futuras llamadas en el *FreeSwitch*.

1.8 Guía de instalación y uso: Aplicación Android

Para poder utilizar la aplicación debemos instalar el archivo ejecutable *.apk*. Para ello, lo pasamos a nuestro terminal móvil a través de cualquiera de las plataformas en las que esté disponible la aplicación y pulsamos sobre él para que se despliegue la instalación.

Cuando la instalación termine se mostrara la pantalla de *login* sobre la que tendremos que pulsar *Create account* para poder tener una cuenta y utilizar la aplicación.

Con la cuenta ya creada debemos *logearnos* en la aplicación con el usuario y contraseña que hemos escogido. Si el usuario y contraseña son correctos pasaremos a la actividad *Navigation*.

Sobre la actividad *Navigation*, la única funcionalidad posible para interactuar con el servidor es a través del botón *doReport*. Dicho botón se encuentra sobre la pestaña *Calls*.

En caso de que hayamos programado llamadas hacia el terminal de la aplicación debemos esperar a que estas concluyan para pulsar el botón *doReport* y que la aplicación envíe al servidor los datos de las llamadas que ha recibido.

La aplicación Android está realizada para comunicarse con un servidor con una IP y puerto fijos. En caso de que cambiemos el servidor de IP y puerto debemos recompilar la aplicación cambiando en el código la IP y el puerto al que debe enviar las peticiones *http* de tipo *get* y *post*. Para ello, abrimos el proyecto y únicamente sobre la clase *NetworkClient* debemos modificar la siguiente línea:

- ```
  public static final String BASE_URL =
  "http://xxx.xxx.xxx.xxx:yyyy";
  ```

Sustituyendo las *x* por la IP que hemos dado al servidor y las *y* por el puerto en el que este escuchando.

Una vez realizada la modificación, suponiendo que estamos sobre la herramienta Android-Studio, debemos darle al botón de *guardar*, y pulsar sobre *Build -> Build Bundle /Build Apk -> Build Apk*. Finalmente, se generara un archivo *.apk* con la aplicación con el código modificado.
  ```

1.9 Planificación temporal

A continuación, se representa en los siguientes diagramas la distribución temporal aproximada que se ha llevado a cabo durante este Trabajo Fin de Grado:

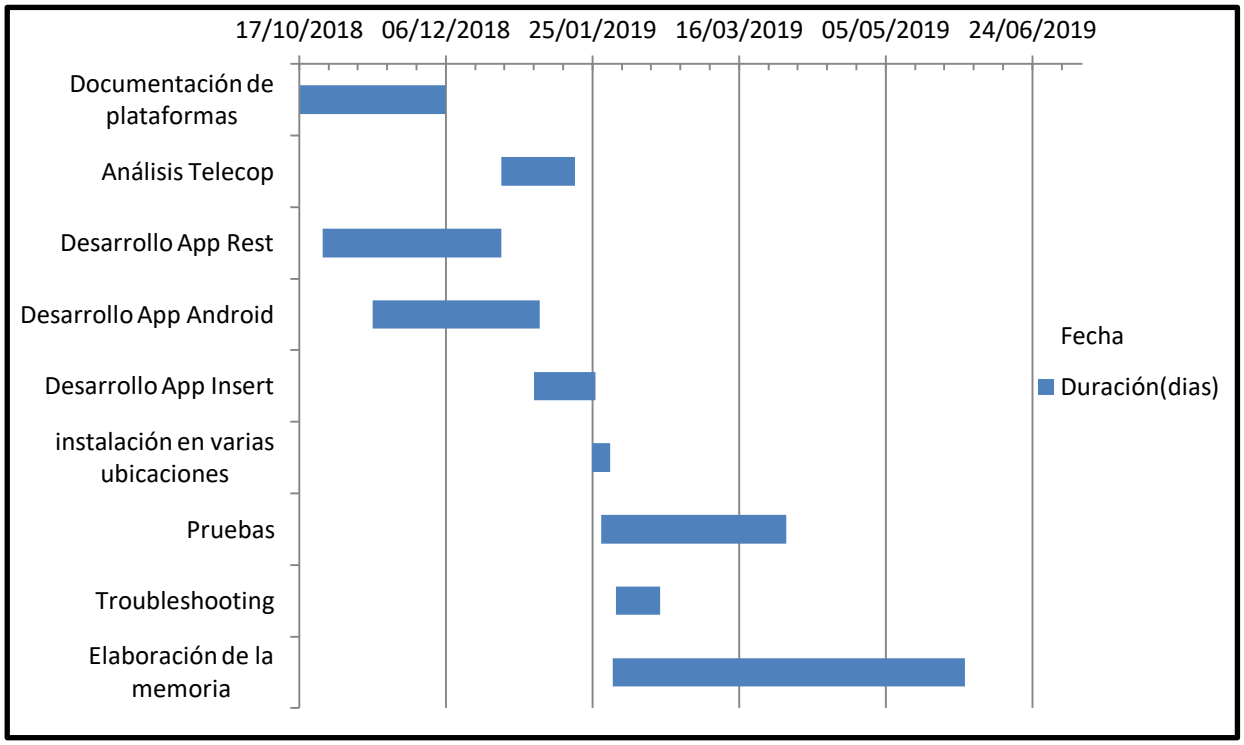


Figura 23: Diagrama de *Gantt*

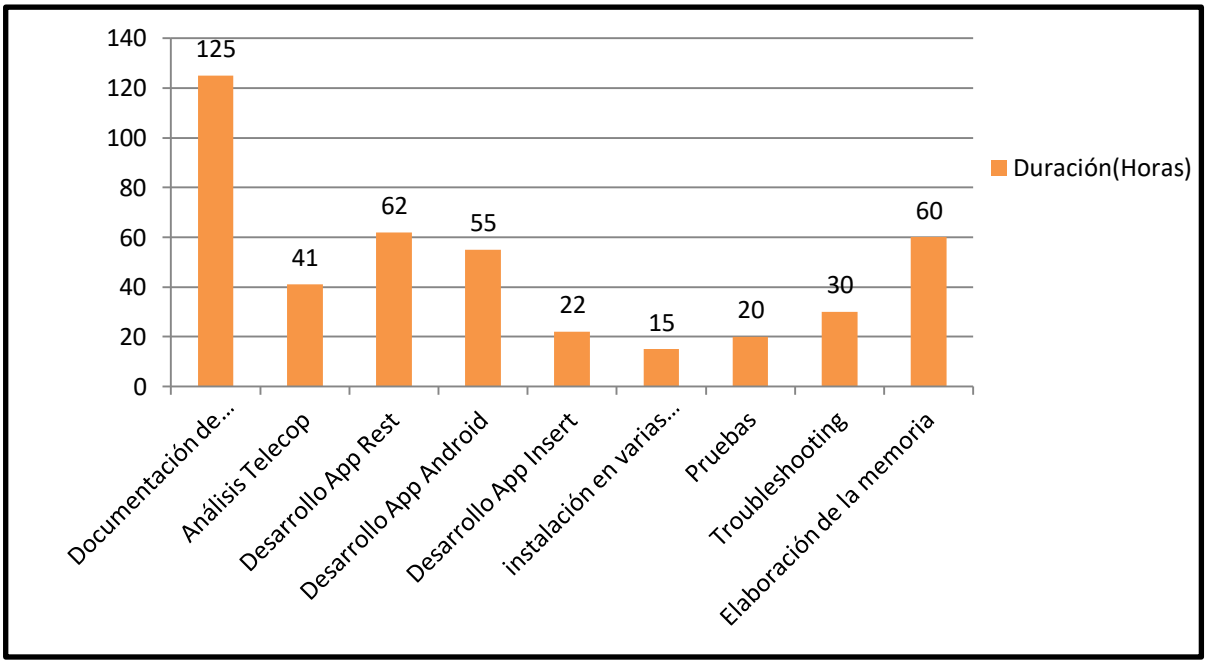


Figura 24: Distribución de horas