ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Desarrollo de servicios de auto-configuración de encaminadores cliente tipo Ethernet, ADSL y de voz usando protocolos basados en DHCP66 y TR069

AUTOR: Sergio Fernández Rubio
DIRECTOR: Francesc Burrull i Mestres
CODIRECTOR: Mathieu Kessler
Septiembre / 2015

Índice general

Índice de figuras

Capítulo 1. Introducción.

En la actualidad, vivimos en una *Era de la Información* (también conocida como *Era Digital* o *Era Informática*) donde las tecnologías de la información y las comunicaciones están avanzando a pasos agigantados. Millones de dispositivos electrónicos se conectan a Internet cada día, y el flujo de información es cada vez mayor, alcanzando a diario cientos de *petabytes* [1].

En medio de esta situación, cada día más usuarios necesitan de los servicios de Internet en todo lugar, haciendo que aumente a grado exponencial el número de dispositivos de red necesarios para satisfacer al cliente. Esto es conocido generalmente como *el Internet de las cosas* (En inglés, *Internet of Things*). Este es un concepto que se refiere al hecho de digitalizar y proveer de conexión a Internet a objetos cotidianos. Más que nunca, en este sentido la revolución está llegando ya para todos los electrodomésticos de casa. Es por ello que se necesitan protocolos de actuación en red, para que satisfagan la ingente cantidad de nuevos usuarios y servicios conectados a Internet.

Primero, en esta memoria explicativa, hablaremos del protocolo DHCP66 y la opción 66 que este protocolo incluye. Veremos cómo se gestiona el provisionamiento del archivo de configuración para los Tenda W308R y para los Grandstream HT502. También veremos cómo se crean estos archivos de configuración en el lenguaje escogido de programación R.

Seguido de esto, hablaremos de la parte del código dedicada a la gestión del alta de nuevo cliente PPPoE, con el fin de proveer de Internet al cliente. Veremos el flujo de datos entre las diferentes bases de datos con las que la empresa trabaja, y los escollos a superar. También estudiaremos la estructura de las diferentes bases de datos.

Por último, explicaremos la incorporación (opcional) de un nuevo ticket de instalación para la posterior gestión por parte de las encargadas de llevar a cabo las instalaciones de nuevos clientes. En esta parte usaremos, junto a R, un script hecho en Python con el paquete Selenium, para así navegar por la web de la base de datos de tickets.

Capítulo 2. Estado de la técnica.

Empezaremos hablando acerca de los métodos de autoconfiguración de dispositivos. Estos se basan fundamentalmente en los que hemos usado en este trabajo, y son el de *DHCP Option 66* y el *Technical Report 069.* Aunque diferentes, ambos tienen en común que son realmente prácticos a la hora de reducir el tiempo de *set-up* para un nuevo router.

2.1. DHCP Option 66

El servicio de aprovisionamiento del DHCP66 sigue un esquema más básico que el de TR069, siendo éste el que a continuación explicaremos.

El sevicio DHCP66 se nutre del servicio DHCP, con el que se obtiene la dirección IP de cualquier nuevo cliente que se conecte a la red. En el mismo paquete que envía el servidor DHCP con la IP del CPE, envía también la opción 66 [2].

Este servidor puede no estar en la misma red local que la del Tenda W308R, y por lo tanto esta petición la haría mediante un *DHCP Relay*. Como segunda opción, el mismo equipo que ejerce como DHCP Relay podría ser el que le suministrara el servicio de DHCP66

Tras la petición de la opción 66, el servidor DHCP le respondería con una IP. Esta IP es la IP del servidor TFTP. Es a esa IP a donde debe ir el router a preguntar por su archivo de configuración. Así que eso hará, y conseguirá autoconfigurarse con ese archivo de configuración que descarga.

2.1.1. Servidores TFTP

Tenemos varias opciones a considerar en cuanto a la elección de provisión de servicios TFTP, todas ellas *Open Source,* tales como *xinetd*, *tftpd32* o *PumpKIN*.

2.1.1.a. Xinetd

Xinetd es un servicio dedicado a administrar la conectividad basada en Internet [3]. Por tanto, es mucho más que un servidor TFTP. Es un servicio diseñado para sistemas UNIX. Sus siglas significan “eXtended InterNET Daemon”. En su versión anterior, llamada *inetd*, contenía menos seguridad que la que implementa xinetd.

Para su correcto funcionamiento, necesitaremos los paquetes auxiliares de xinetd: *tftpd* y *tftp*.

2.1.1.b. Tftpd32

Ejecutable de Windows, que incluye no sólo un servidor TFTP, sino que incluye servidores DHCP, DNS y SNTP, como también un cliente TFTP [4].

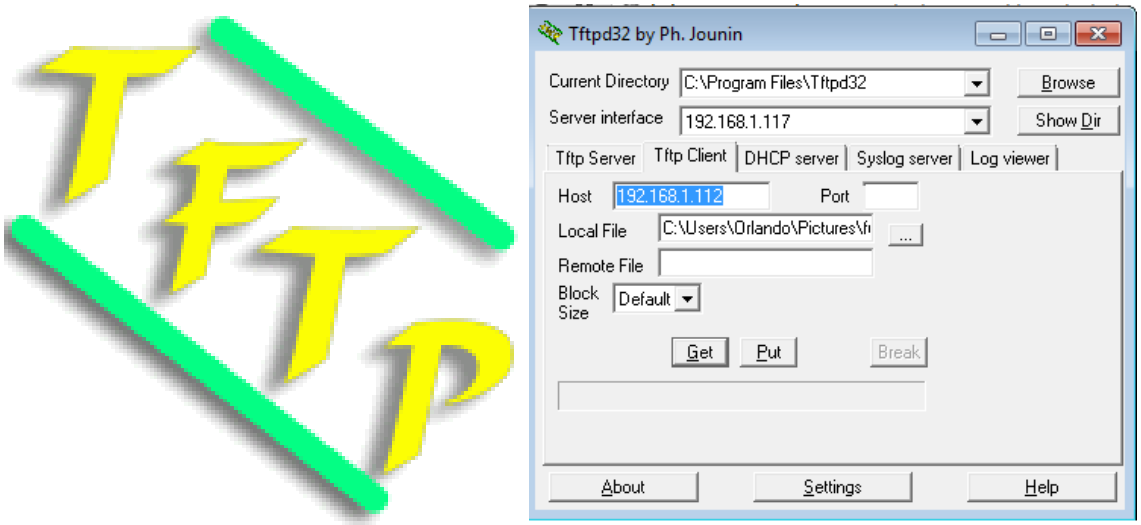

Figura 1: Tftpd32 logo y vista principal de la GUI

2.1.1.c. PumpKIN

Disponible tanto para Windows como para OSX, incluye tanto servidor TFTP como cliente TFTP [5].

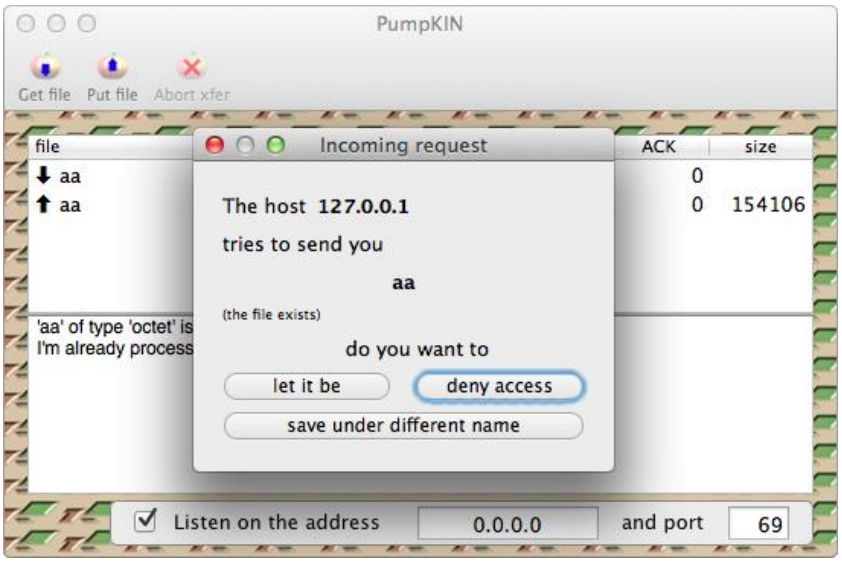

Figura 2: GUI de PumpKIN para OSX

2.1.2. Servidores HTTP

Los servidores HTTP se definen como servidores que procesan peticiones vía HTTP. Desde que surgió Internet, los servidores web no han parado de aumentar en tamaño, número, complejidad y funcionalidad, como vemos en la siguiente comparativa [6]:

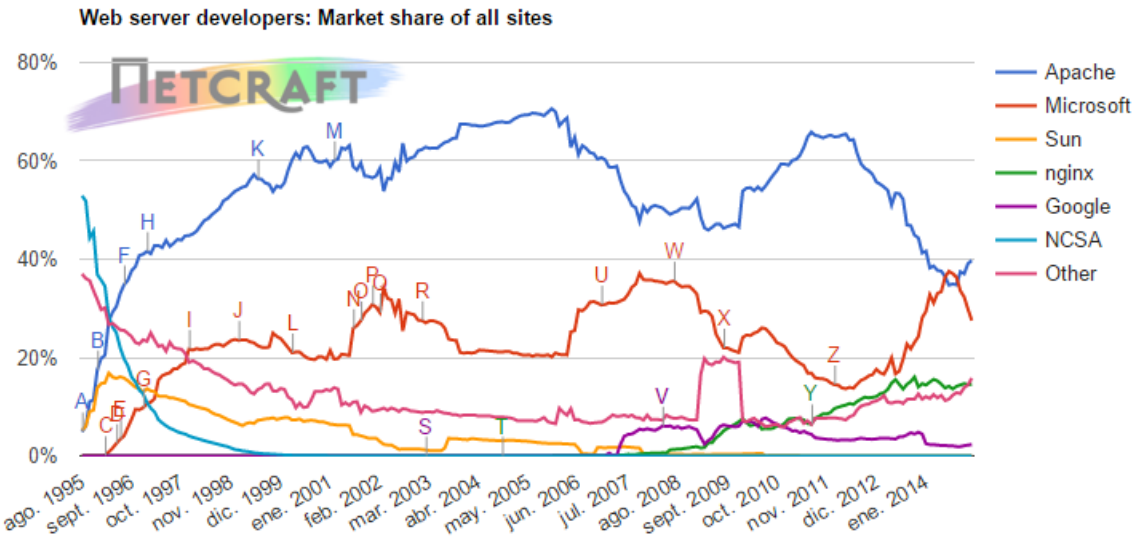

Figura 3: Comparativa de la cuota de mercado de los servidores web

En nuestro desarrollo, analizaremos los servidores Apache y Nginx, los grandes Open-Source de hoy en día.

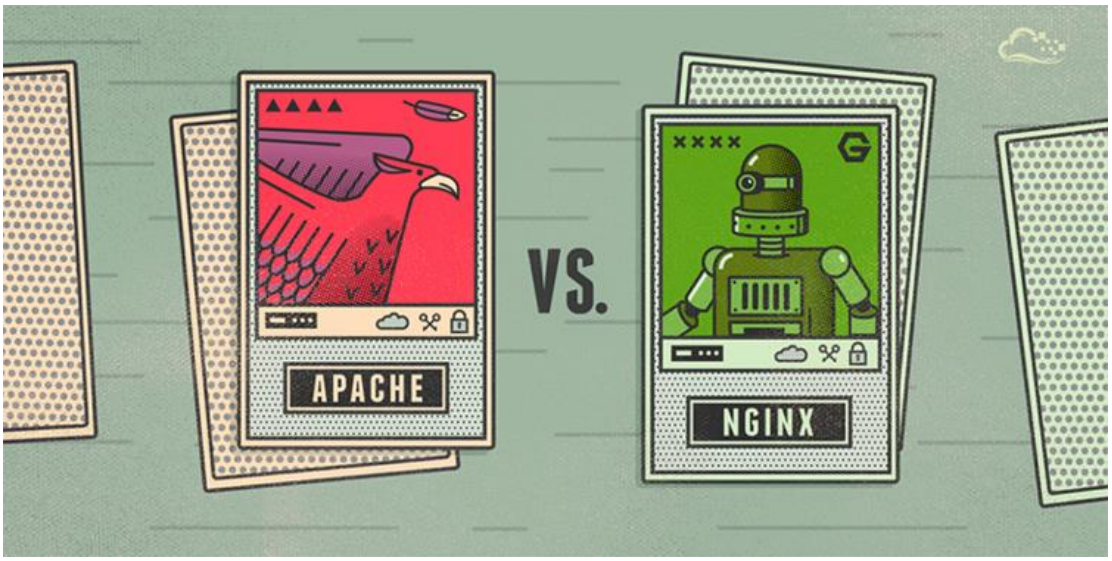

Figura 4: Apache vs Nginx

2.1.2.a. Apache HTTP Server

El proyecto Apache HTTP Server es un esfuerzo conjunto de desarrollar y mantener un servidor HTTP Open-Source para sistemas operativos modernos, incluyendo Unix y Windows [7].

Figura 5: Logo de Apache HTTP Server

Nacido en 1995, el objetivo del proyecto es proveer un servidor extensible, eficiente y seguro, en sincronización con el estándar HTTP.

Es el servidor más usado en todo el mundo. La mayor diferencia entre Apache y Ngnix es la manera de manejar las conexiones y el tráfico. Apache usa módulos de multiproceso (MPM), que dictan cómo las peticiones de los clientes son manejadas. Esto hace que los administradores manejen la arquitectura de gestión de las conexiones de manera sencilla.

2.1.2.b. Nginx

Con la concurrencia en mente, nginx nació en 2004 por manos de Igor Sysoev, en respuesta al problema C10K, que era un reto para los servidores web que signficaba gestionar 10.000 conexiones concurrentes como mínimo [8].

Figura 6: Logo de Nginx

Es mayormente usado en los servidores con gran actividad de datos. En su parte de gestión de las peticiones, se difiere en Apache en que nginx crea varios procesos, en los cuales, individualmente, pueden manejar miles de conexiones. Esto se consigue implementando un mecanismo de *loop* muy rápido que continuamente supervisa los eventos de proceso.

Estos eventos dentro del *loop*, son procesados asincrónicamente, permitiendo que el trabajo se realice de una forma no-bloqueante.

2.2. Technical Report 069 (TR-069)

Publicado por el *Broadband Forum* en el año 2004, su misión es configurar remotamente los CPE (*Customer Premises Equipment*) a través de un ACS (*Auto-Configuration Server*) [9]. También conocido como CWMP (*CPE WAN Management Protocol*), este protocolo es diferente en su filosofía al del DHCP66, adoptando un estilo orientado a comandos (o parámetros) más que a la simple descarga de un archivo de configuración, aunque esto último también lo incluye en su especificación.

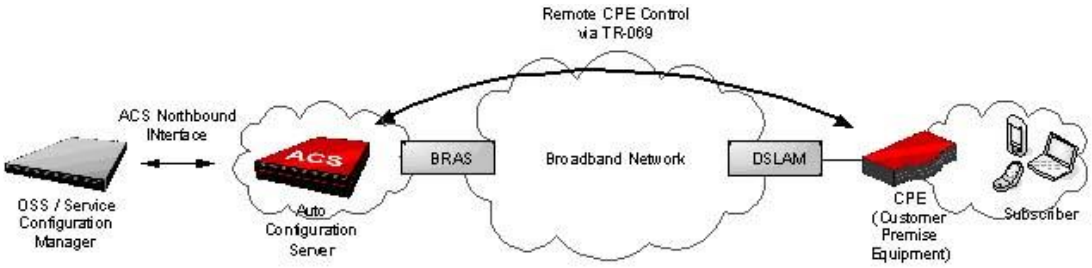

Figura 7: Ejemplo de despliegue de TR-069

Se basa en unos protocolos SOAP/HTTP (Intercambio de mensajes con esquema SOAP basada en el conocido protocolo HTTP), en el que estos parámetros son enviados. Estos parámetros son el modelo de datos con el que el ACS trabaja. Como estándar, este modelo de datos tiene un prefijo en común: *Device* e *InternetGatewayDevice.* Cualquier fabricante puede elegir cuál usar, pero debe ser consistente al usarlo.

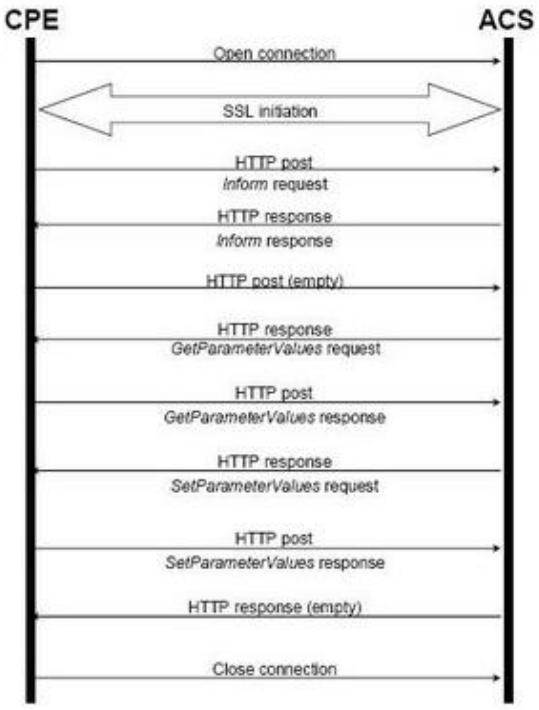

Figura 8: Ejemplo de transacción entre CPE y ACS

Como servicios generales del TR-069 tenemos:

- Gestión y configuración automática de los servicios en un CPE
- Configuración remota de un CPE
- Gestión de firmware
- Gestión de versiones
- Actualización de la gestión y de control de la ejecución
- Log de análisis dinámico y mensajes
- Diagnósticos
- Conectividad y control de servicios

Pasaremos, a continuación, a hablar del estado de la técnica en servidores de autoconfiguración.

2.2.1. ACS

Tenemos mucha variedad de soluciones software para el correcto funcionamiento del ACS (Auto-Configuration Server), pero muchas de ellas son proyectos descontinuados. Hablaremos de los que están actualmente en desarrollo, tanto Open Source como privativos.

2.2.1.a. GenieACS

Proyecto de Zaid Abdulla, con la mayor comunidad de usuarios de entre todos los proyectos Open Source. Su principal objetivo es el rendimiento y la optimización, usando medios concurrentes y nuevas tecnologías como *Node.js*, *Redis*, y *MongoDB*. También tiene una modalidad de pago [10].

Figura 9: Logo de GenieACS

Diseñada también para facilitar el diseño de nuevas aplicaciones, forma parte de un repositorio en GitHub [11] y existen varios *forks* que el autor aprueba y recomienda [12]. Además, consta de una API muy completa y accesible para establecer parámetros de un CPE, presets estáticos, hacer búsquedas de CPEs y borrado de elementos.

Consta de una GUI que desarrolla en *Rails*, también cuenta con 3 motores, cada uno encargado de tres puertos por los que los CPE se comunican con el ACS, implementado en *Node.js*, y dos bases de datos, una *MongoDB,* que es la encargada de guardar toda la información necesaria de cada router conectado, y otra *Redis* que es la encargada de guardar los datos de la capa de caché.

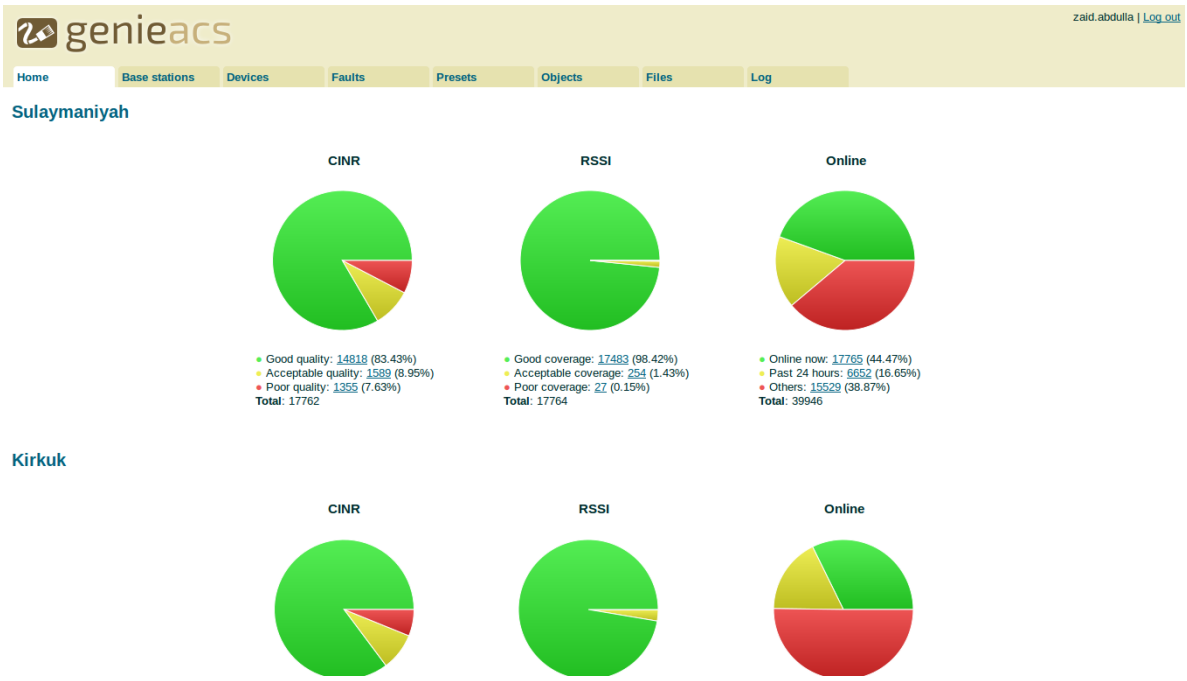

Figura 10: Gráficos de monitorización en la pantalla principal de GenieACS

2.2.1.b. Easycwmp

Es el único Open Source que actualmente está soportando la plataforma OpenWRT. Proveniente del proyecto (ya no mantenido) *freecwmp*, está siendo desarrollado por PIVA Software. Su objetivo es ser fiel al estándar TR-069 [13].

Figura 11: Logo de EasyCwmp

El diseño de EasyCwmp incluye dos partes:

- El núcleo de EasyCwmp: Contiene el motor de comunicación con el ACS, que está siendo desarrollado en C.
- Los scripts de EasyCwmp: Son una colección de scripts que ejecutan métodos del ACS (get, set, add, delete, apply setting, download file, upgrade firmware, reboot…) que están siendo desarrollados en script *ash.*

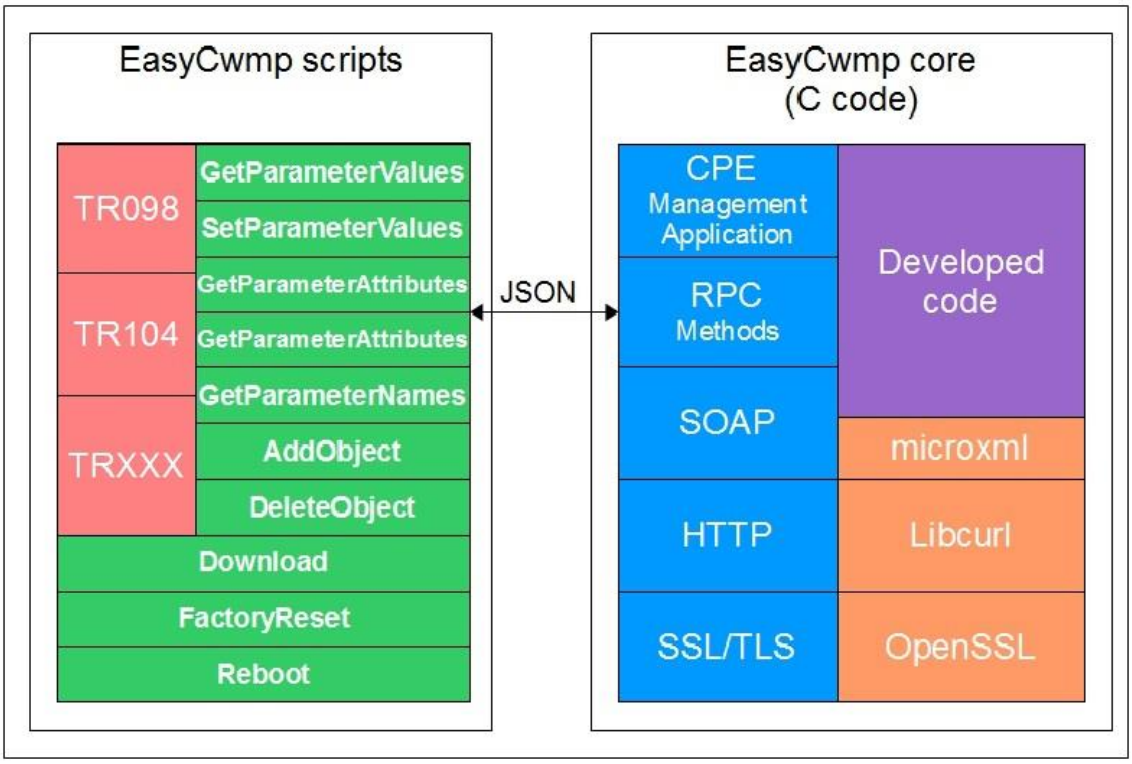

Figura 12: Esquema de funcionamiento de EasyCwmp

Otro objetivo que quiere cumplir EasyCwmp es separar el motor de EasyCwmp escrito en C, del script en ash, para que añadir nuevas funcionalidades sea sencillo.

2.2.1.c. AXESS.ACS

Principal producto de la empresa Axiros. Es un software privativo, pero muy avanzado y dispuesto a superarse a sí mismo [14].

AXESS.ACS
Open Device Provisioning

Figura 13: Logo de AXESS.ACS

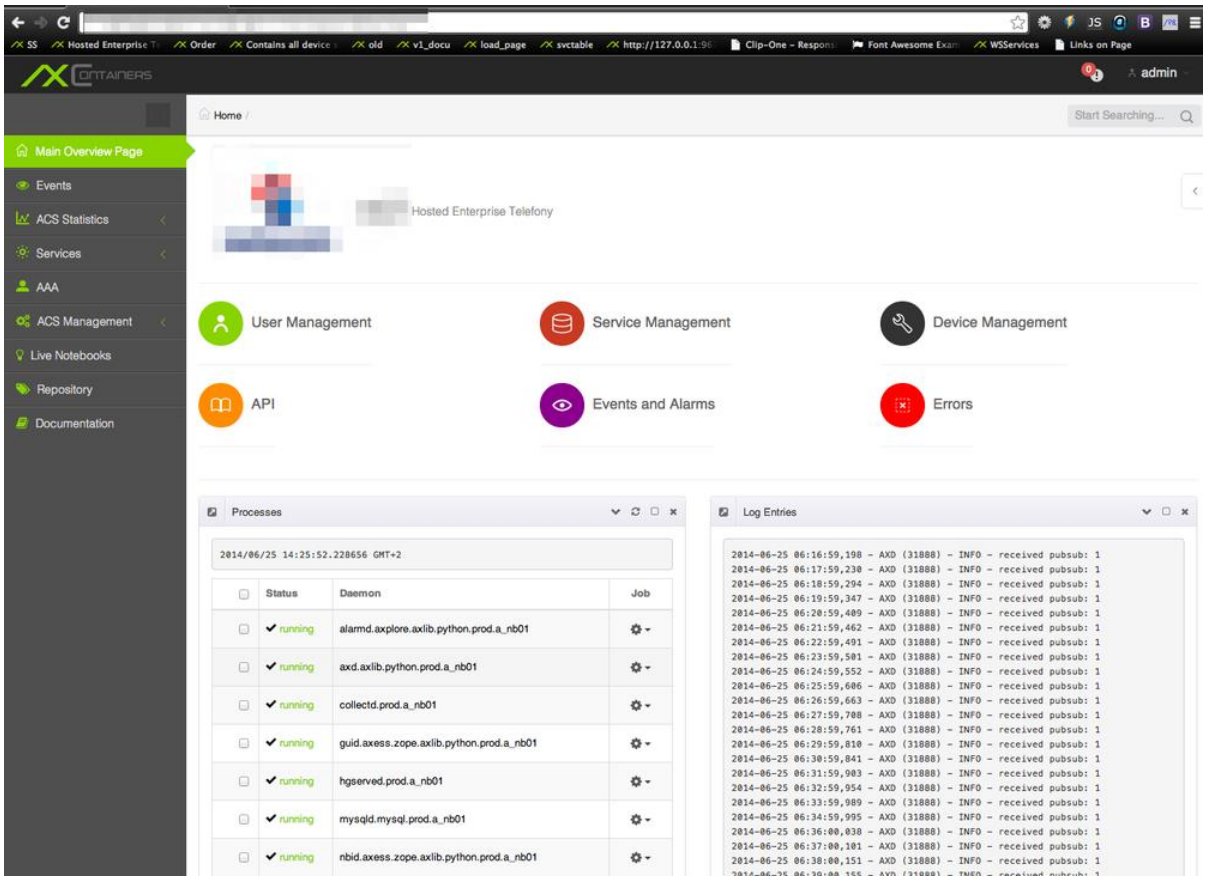

Figura 14: Pantalla principal de AXESS.ACS

Cuenta con las siguientes funcionalidades, según la misma empresa:

- Mantenimiento remoto de CPEs
- Aprovisionamiento instantáneo y sin necesidad de configurar paso a paso el router
- Rápida monitorización, alta tasa de muestreo
- En completa conformidad con el estándar TR-069
- Portal de servicio para el cliente tipo *self-service*
- Gestión del firmware inteligente
- Gestión de la seguridad
- Múltiples protocolos NBI, extensibles
- Interoperabilidad
- Permisos para añadir, ver, cambiar y borrar
- Dispone servicios de email, syslog, XML-RPC, SOAP, clientes JMS
- Más de 100 lenguajes
- Sistema de autorización customizable
- Operaciones en masa programadas
- Habilidad de operaciones multi-step:
- Instalación rápida en sistemas Linux como un paquete
- Soporta también Solaris

- Servidor de aplicación incluyendo bases de datos SQL
- Posibilidad de agrupar jerárquicamente por CPEs

Cuenta con grandes socios como ZyXEL, una empresa de fabriación de routers, principalmente en tecnología DSL

2.2.1.d. Unified Device Management Platform de AVSystem

Principal producto de la empresa AVSystem. Es un potente servicio de autoconfiguración basado en TR-069, que promete autoconfigurar todo tipo de dispositivo, de cualquier fabricante, con cualquier proocolo, en un gran número de dispositivos [15].

Figura 15: Logo de AVSystem

Dispone de las siguientes funcionalidades principales:

- Activación automática de servicio
- Portal para clientes
- Módulos autoadministrados
- Reportes y monitorización proactiva
- Arquitectura basada en tareas y organización en grupos
- Interfaz unificada
- Soporte para dispositivos IoT, M2M y servicios basados en nube
- Servicio de localización geográfica

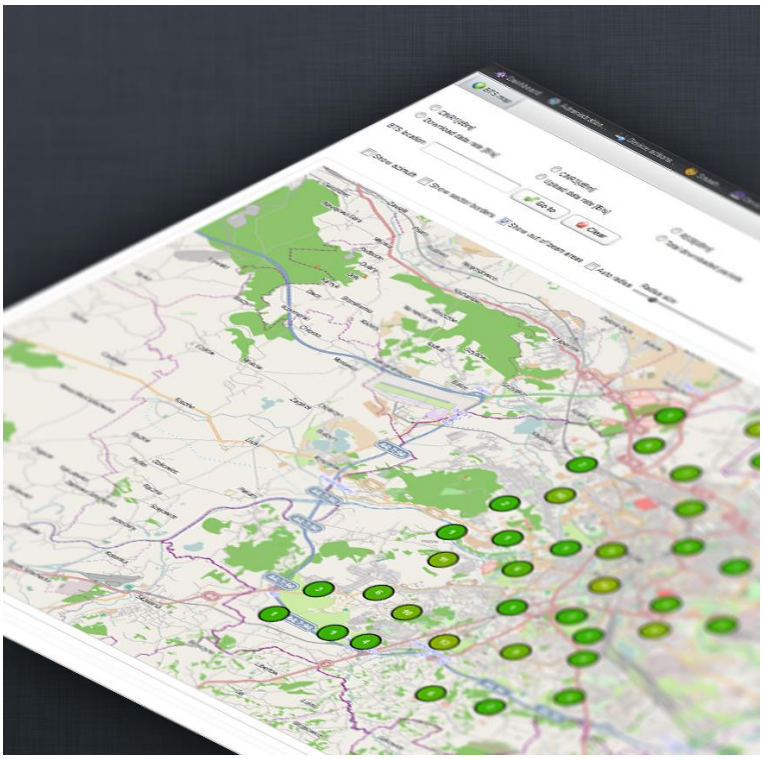

Figura 16: Vistazo de la aplicación en su modo de localización de dispositivos

Disfruta de grandes socios como SagemCom, Vodafone, Netia, AlbTelecom o Grandstream.

2.3. Lenguajes de programación

El abanico de lenguajes de programación para desempeñar la tarea de crear los archivos de configuración, proveer de Internet al cliente, e insertarlo en la base de datos de tickets es muy amplio. Es por ello que intentaremos resumir las diferentes opciones que tendremos a nuestra disposición, y más adelante, en la explicación de la tarea, explicaremos por qué hemos elegido este/estos lenguajes.

2.3.1. R

R es un lenguaje de programación orientado al análisis y procesamiento de datos. Es un Proyecto GNU, similar al lenguaje S (de donde provino). Este último, S, fue desarrollado en el antiguo *Bell Laboratories* por John Chambers y sus compañeros. R nació a partir de R Robert Gentleman y Ross Ihaka, miembros del Departamento de Estadística de la Universidad de Auckland, en Nueva Zelanda [16].

Figura 17: Logo de R

R es un lenguaje Orientado a Objetos, y bajo este complejo término se esconde la simplicidad y flexibilidad de R. La sintaxis de R es simple e intuitiva, con lo que trabajar con él se hace fácil una vez que te acostumbras. R parte de lenguajes de bajo nivel, esencialmente C y Fortran.

Orientado a Objetos significa que las variables, datos, funciones o resultados se guardan en la memoria activa del computador en forma de objetos con un indicador específico. El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos).

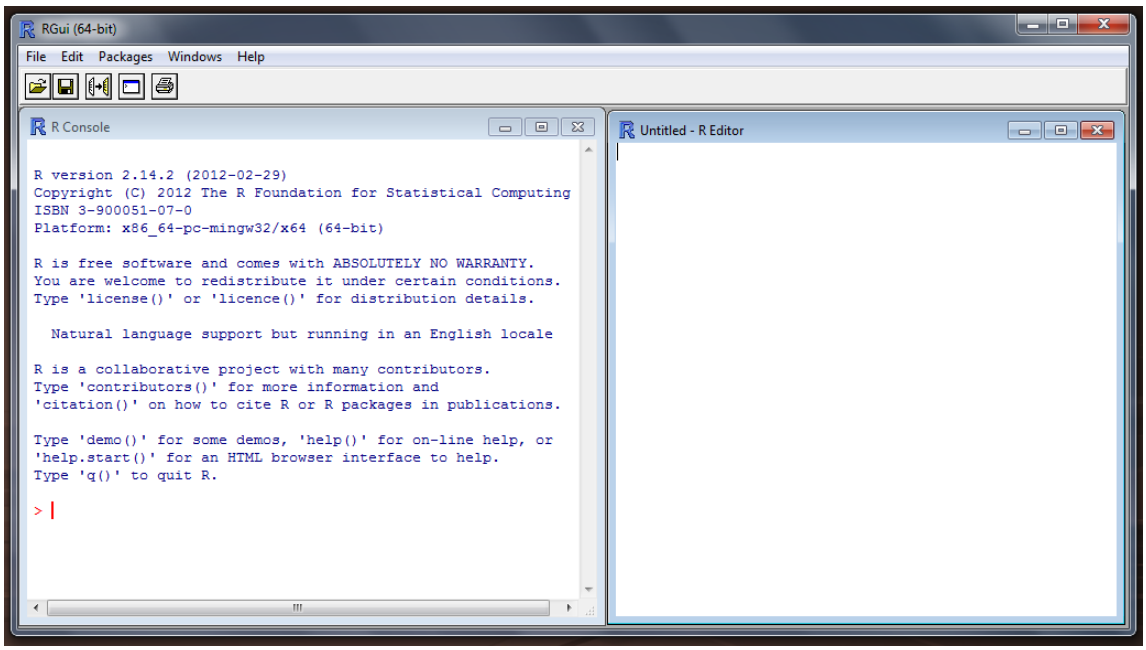

Figura 18: Pantalla principal de RGui para Windows

R es un lenguaje interpretado, no compilado, lo cual es muy útil a la hora de ejecutar scripts, sin tener que compilar todo el código. Como facilidades que incluye, según la misma R Foundation [17], son las siguientes:

- Optimizado y efectivo manejo de datos y de almacenamiento
- Una *suite* de operadores para trabajar con *arrays*, particularmente matrices
- Una colección integrada de herramientas para el análisis de datos
- Herramientas gráficas para el análisis de datos
- Un buen desarrollado, simple y efectivo lenguaje de programación que incluye

condicionales, iteradores, funciones diseñadas por el usuario, y herramientas de entrada y salida.
- IDE desarrollado enteramente para R: RStudio

Como contra, tiene la pega de que usa muchos recursos, y es más ineficiente, comparativamente hablando, que otros lenguajes de programación. Por eso se han creado paquetes para R que recodifican parte del código, usando R, para hacerlo más veloz. Ejemplos son pqR, renjin, FastR, Riposte, RevoScaleR (comercial) y Foreach (comercial).

La gran ventaja sobre los demás lenguajes es la facilidad del tratamiento de la información, el análisis de datos, y la incorporación de nuevos paquetes (o librerías) para el uso dentro del sistema. También el repositorio de paquetes R es mucho más grande que las de los demás lenguajes. Los más usados y conocidos son dplyr, plyr, data.table, stringr, zoo, ggvis, lattice, ggplot2, caret… entre muchos otros.

2.3.2. Python

Python es un lenguaje interpretado de alto nivel, con semántica dinámica [18]. Es también un lenguaje con orientación a objetos, como ya hemos explicado antes en el apartado en el que hemos hablado de R. Es también un lenguaje en el que la sintaxis es sencilla, aunque no tanto como la de R, pero es por ello un lenguaje más potente para su uso en entornos complejos y costosos en cuanto a uso de recursos.

Figura 19: Logo de Python

Python es un lenguaje multiuso, nacido en 1991 de la mano de Guido Van Rossum, no dependiendo de ningún lenguaje auxiliar. Su objetivo se centra en la productividad y legalidad del código. Tiene un buen apoyo por parte de la comunidad en sitios como Stackoverflow, en listas de correo y documentación base.

Es de resaltar que es uno de los pocos lenguajes en los que la indentación del código realmente tiene efecto en la ejecución. El debugging tiene por pro en Python que el código escrito es sencillo de debugear debido a herramientas internas del lenguaje.

Python, en comparación a R, tiene una curva de aprendizaje menor, y es por esto que es un buen lenguaje para principiantes en programación.

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```

Figura 20: Ejemplo de código en Python
Observamos su indentación y la inexistencia de corchetes

Hay diferentes IDEs para trabajar con Python, entre ellos Spyder, IPython Notebook, Roden, PyCharm, WingIDE, PyDev… Sus librerías más famosas incluyen pandas, requests, scrapy, wxPython, NumPy, SciPy, Selenium… entre muchos otros.

2.3.3. Java

Java es un lenguaje interpretado, fuertemente tipado, y de más bajo nivel que R o Python [19].

Figura 21: Logo de Java

Nacido en 1995 por manos de James Gosling, que trabajaba en Sun Microsystems. La sintaxis del lenguaje deriva mucho de C y C++, pero es un lenguaje de mayor nivel que éstos últimos.

Los IDEs más famosos para trabajar con Java son Eclipse, NetBeans, IntelliJ, Android Studio, BlueJ, JDeveloper…

La clara desventaja frente a los demás lenguajes es que para escribir un mismo código

en Java que en Python (o R) se necesitan muchas más líneas de código. Esto se hace evidente cuando queremos leer o escribir datos en disco. En Python o R, es una simple línea, mientras que Java, aparte de que tiene muchas formas de leer/escribir a disco, necesita entre 10 y 20 líneas [20].

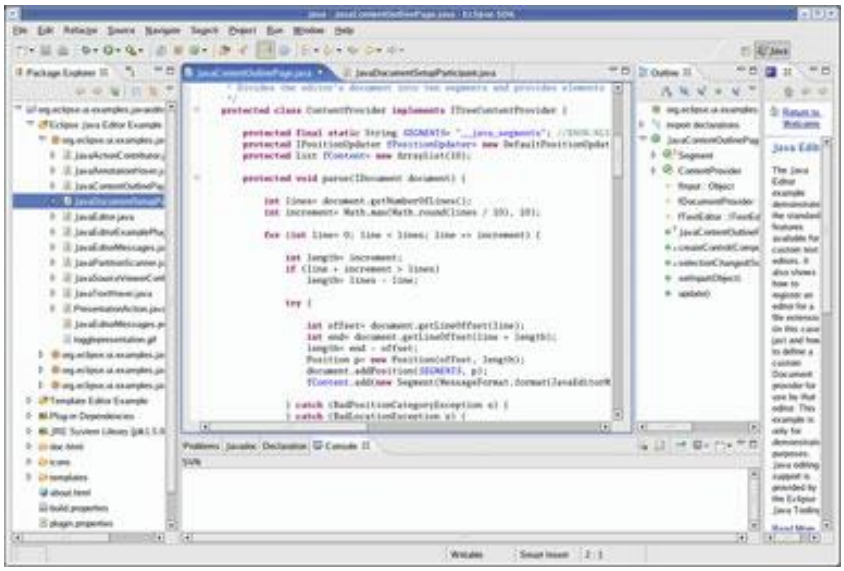

Figura 22: Entorno Eclipse de desarrollo Java

La ventaja principal de Java es la interoperatividad entre distintos sistemas operativos y la velocidad de ejecución, que en muchos casos es más rápida que en Python o R. También tiene ventajas en cuanto al manejo de la concurrencia, ya que contiene muchas herramientas nativas para su fácil manejo.

2.4. Markdown

Markdown es un lenguaje de marcado ligero creado por John Gruber y Aaron Swartz, que trata de conseguir la máxima legibilidad y facilidad de publicación en Internet, usando texto plano [21]. El formato de salida son documentos XHTML bien formados. Fue escrito originalmente en Perl, pero ha sido escrito en multitud de lenguajes de programación, incluyendo PHP, Python, Ruby, Java, Common Lisp, R…

No existe ningún estándar creado más allá del que publicó Gruber en su primera versión en Perl, y por eso importa relativamente poco en la tecnología que se use, usándose la más cómoda para el programador (normalmente, según el lenguaje utilizado)

2.5. Bases de Datos

Más y más información es generada por todo el mundo cada día. La cantidad de información que se genera cada día es del orden de los Exabytes (que equivale a 1 000 000 000 000 000 000 de bytes). Es de esperar que los sistemas de recolección de

datos sean potentes y ágiles a la hora de manejar esos datos, y por eso aquí investigaremos superficialmente las tecnologías principales a la hora de elegir una base de datos.

2.5.1. Soluciones SQL

Es el gran conocido amigo de las bases de datos. Sus siglas significan Structured Query Language, y es un lenguaje declarativo de acceso a bases de datos relacionales. Fue creado en el 1974 por Donald D. Chamberlin mientras trabajaba en IBM [22].

Figura 23: SQL

Aunque SQL es un estándar que ha ido evolucionando en el tiempo, este es voluntario, que quiere decir, que cada empresa puede crear una sintaxis diferente basándose en el estándar actual. Por ello, vamos a citar varios ejemplos de soluciones SQL:

2.5.1.a. MySQL

Creada por MySQL AB, una compañía sueca, y mantenida por Oracle Corporation, es una base de datos relacional Open Source, que en Julio de 2013 alcanzó el puesto número dos en popularidad mundial [23]. Es el componente central en aplicaciones web LAMP (Linux, Apache, MySQL, Perl/PHP/Python).

Figura 24: Logo de MySQL

MySQL está escrito en C y C++, y el parser SQL está escrito en yacc, aunque usa un analizador léxico propio de MySQL. Es multiplataforma, y tiene un extensivo manual a disposición en su web, donde se pueden encontrar toda la teoría necesaria y también toda clase de ejemplos prácticos. Además, se puede obtener soporte vía Stackoverflow, canales IRC y en demás foros, así como un soporte de pago.

Como funcionalidades, incluye:

- Soporte inter-plataforma
- Amplio subconjunto del lenguaje SQL
- Posibilidad de selección de mecanismos de almacenamiento
- Transacciones y claves foráneas
- Conectividad segura
- Replicación
- Búsqueda e indexación de campos de texto

Aunque originalmente no fuera tan robusto como otros lenguajes SQL, se ganó a los programadores por su facilidad y simpleza, y fue más adelante cuando completaron todas las operaciones más complejas de los sistemas SQL, tales como las transacciones.

2.5.1.b. PostgreSQL

Base de datos Open Source, multiplataforma [24]. No es manejado por ninguna empresa, sino que es dirigido por una comunidad de desarrolladores. Tiene más de 15 años de edad, y su arquitectura ha sido exhaustivamente probada y revisada.

Figura 25: Logo de PostgreSQL

Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema, ya que un fallo en uno de los procesos no afectará el resto.

Se descompone entre los siguientes módulos:

- Aplicación cliente: Es la aplicación cliente que utiliza como administrador de bases de datos. La conexión ocurre vía TCP/IP o sockets locales.

- Demonio postmaster: Es el proceso principal de PostgreSQL. Escucha por un puerto o socket, para establecer conexiones entrantes de clientes. También se encarga de crear procesos hijos que explicaremos más adelante.

- Ficheros de configuración: Los 3 principales son *postgresql.conf*, *pg_hba.conf* y *pg_ident.conf*.

- Procesos hijos: Se encargan de autenticar a los clientes, gestionar sus consultas y mandar los resultados a las aplicaciones clientes.

- Memoria compartida caché: Usada por PostgreSQL para almacenar datos en caché.

- Write-Ahead Log (WAL): Componente encargado de asegurar la integridad de los archivos.

PostgreSQL ha sido un gran proyecto durante años por su capacidad para autogestionarse mediante una comunidad amplia y de diversas culturas. Según ciertas estimaciones por parte de COCOMOII, para la versión 9.0.0 de PostgreSQL el coste total del proyecto asciente hasta aproximadamente los 63 millones de dólares. Y recordamos, es un proyecto Open Source.

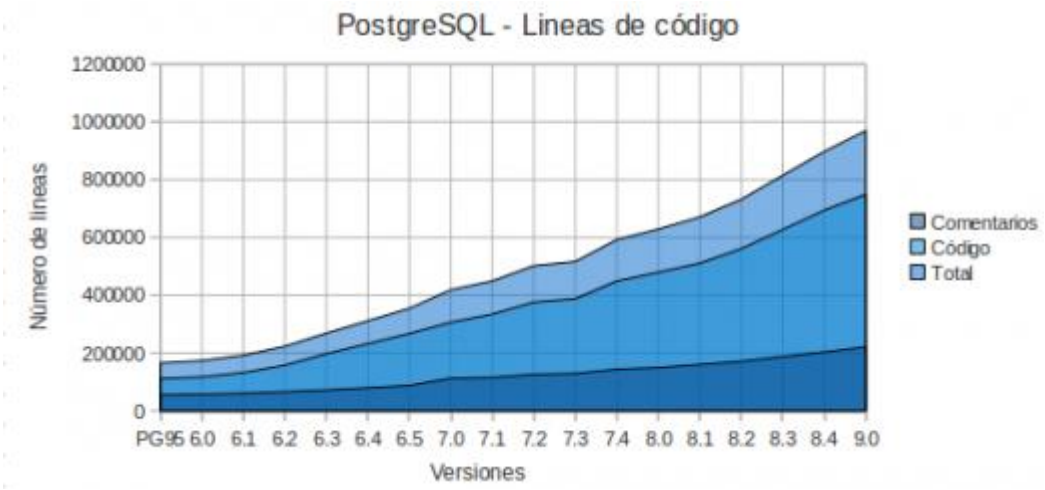

Figura 26: Líneas de código frente a versión de PostgreSQL.
Se observa claramente el gran crecimiento que ha tenido

2.5.1.c. SQLite

Creada por D. Richard Hipp, es una base de datos Open Source creada en agosto del 2000. Es multiplataforma, y es compatible con ACID [25].

Figura 27: Logo de SQLite

Es la base de datos más usada en el mundo. Y es de resaltar también que está contenida en una relativamente pequeña (~275 kiB) biblioteca escrita en C. Y contrario a lo comúnmente encontrado en otras soluciones SQL, no es una base de datos basada en cliente-servidor, sino que está embedida en el programa final.

¿Por qué es la base de datos más usada en el mundo? Porque la podemos encontrar en:

- Todos los dispositivos Android
- Todos los dispositivos iOS
- En Firefox, Chrome y Safari
- En cada cliente Skype
- En cada cliente iTunes
- En cada cliente Dropbox
- En cada cliente TurboTax y QuickBooks
- En la mayoría de sets de televisión y dispositivos reproductores
- En la mayoría de servicios multimedia para automóviles
- Muchas innumerables más aplicaciones

Y también es la más usada por estos distintivos rasgos:

- No hace falta prácticamente set-up. No necesita ninguna instalación, y no se necesita activar ningún proceso. No hace falta siquiera agregar usuarios. ¿Por qué? Porque no hay ningún archivo de configuración que configurar, y el sistema está embedido en la aplicación.

- No existe un servidor, ni un proceso del cual depender. Los programas que quieran acceder a la base de datos, se comunican con el servidor usando un tipo de comunicación entre procesos, usualmente basado en TCP/IP. No pasan por ningún servidor, simplemente leen y escriben en los archivos de la base de datos que están en disco.

- Compacta. Cuando está optimizada, con todo activado, ocupa menos de 500KiB en tamaño. Incluso si no necesitamos ciertas funcionalidades, el tamaño puede bajar hasta los 300KiB.

- Estable en todas las plataformas. Un mismo archivo de base de datos en una máquina de 32 bits, Little-endian, con Linux, funciona igual que en una máquina de 64 bits, Big-endian, con Windows.

- Valores de tamaño variable. Es de señalar que cuando creas una columna en la base de datos, si por ejemplo la creas como un tipo VARCHAR(100), si en la primera fila de esa columna creas un carácter de longitud 5, en disco sólo se guardarán esos 5 caracteres que has introducido, no 100. Esto es lo contrario a lo que hacen la mayoría de bases de datos.

2.5.2. Soluciones NoSQL

¿Por qué NoSQL? NoSQL nació a partir de compañías como Google, Facebook, Amazon y LinkedIn, para superar las limitaciones de las bases de datos relacionales (SQL) para el uso de modernas aplicaciones web. Está ampliamente relacionado con las nuevas tecnologías de Big Data [26].

Figura 28: Familia de bases de datos NoSQL

No hace mucho, 1.000 usuarios para una aplicación eran muchos, y 10.000 se estimaba como caso extremo. Pero en poco tiempo, hasta llegar a hoy, 3.000.000.000 personas se conectan a internet y pasan pasan online sobre unas 35.000.000.000 horas al mes, y esto seguirá creciendo día a día.

Esquemas relacionales y no relacionales son muy diferentes. En el modelo relacional separa los datos en muchas tablas interrelacionadas, pero conforme se va agrandando la base de datos, la información que se debe presentar a la aplicación tiene que ser recolectada de muchas tablas. Por el contrario, el modelo NoSQL agrega contenidos de diferentes documentos y las presenta juntas. A pesar de que esto gasta más memoria, ya no hay esta restricción porque el modelo NoSQL permite múltiples nodos donde alojar las partes de la base de datos.

2.5.2.a. MongoDB

Nacida en 2009 por la compañía de software 10gen, es un sistema de bases de datos Open Source y multiplataforma orientado a documentos [27]. Es la solución líder de bases de datos NoSQL. El formato en que MongoDB guarda estructuras es dinámico, y lo llama *BSON* (Binary JSON).

Figura 29: Logo de MongoDB

¿Por qué es tan popular MongoDB?

Básicamente, por su agilidad. Antes de existir este tipo de bases de datos, era aceptable pasarse meses planeando una aplicación y su esquema de datos asociado, desarrollándolo, y tras esto, nunca más se modificaría. Vivimos en la era de la *Big Data*, y debemos constantemente reinventar nuestras aplicaciones mientras conseguimos más y más datos.

También, es muy popular porque es muy sencilla de aprender y de manejar. Tiene una estructura fácil de escalar, y la misma empresa ofrece cursos de formación para todo tipo de profesionales y de usos: La mongoDB University.

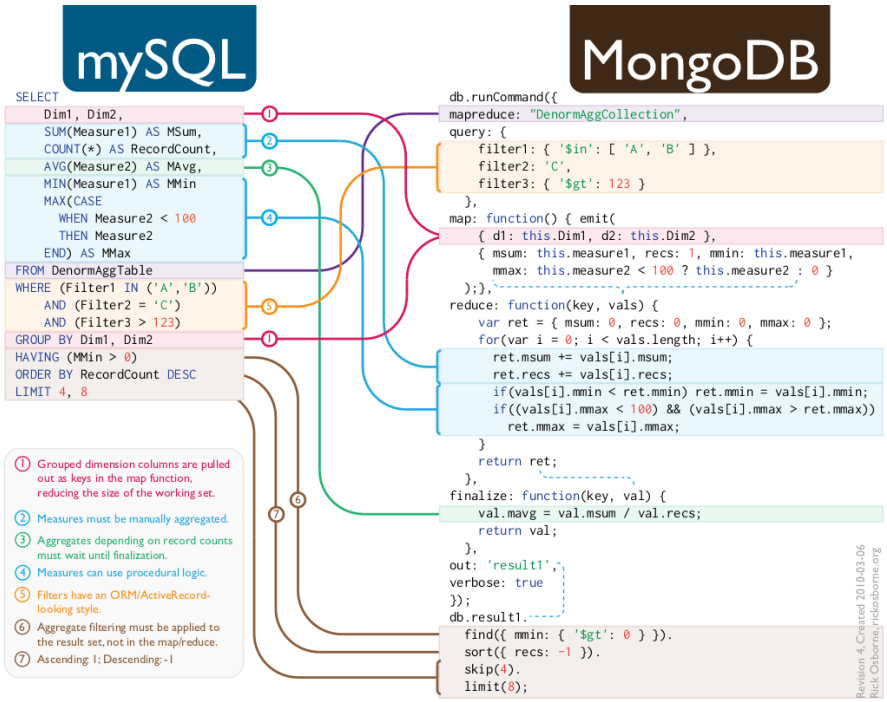

Figura 30: Interesante comparativa entre una consulta en MySQL frente a la misma en MongoDB

2.6. Routers

También conocidos como enrutadores, o encaminadores de paquetes, es un dispositivo que proporciona conectividad a nivel de red (o nivel tres, según el modelo OSI).

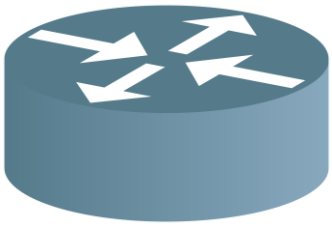

Figura 31: Modelo tradicional de un router

Para ser considerado un router, debe conectar dos o más líneas de datos de diferentes redes. Cuando un paquete llega por uno de los puertos de entrada, el router lee la dirección del paquete para determinar por qué salida deberá salir, usando una tabla de enrutamiento o unas políticas basadas en protocolos de enrutamiento.

Aunque hay muchas marcas de fabricantes de routers, hablaremos muy brevemente de dos principales en el mundo del networking:

2.6.1. Cisco Systems Inc.

Cisco Systems, Inc. es una multinacional americana, centralizada en San Jose, California [28]. Nacida en 1984 por Leonard Bosack, y su mujer Sandy Lerner, diseña, fabrica y vende equipos de red. Es el líder en soluciones de redes para Internet, ya que el 85% del tráfico global pasa por dispositivos Cisco.

Figura 32: Logo de Cisco

Cisco, para incentivar el uso de su tecnología y para certificar a los conocedores de ella, creó los llamados Cisco Careers, que son exámenes dirigidos a los profesionales del sector, en cinco diferentes niveles: Recién llegado, socio, profesional, experto y arquitecto. Cada una de las ramas está subdividida en varias especializaciones.

2.6.2. MikroTik Ltd.

MikroTik Ltd. es un fabricante letón de equipos de redes [29]. Fundada en 1995, se diferencia fundamentalmente por ser una alternativa económica a los caros dispositivos de red actuales.

Figura 33: Logo de MikroTik

Es notable destacar también que, aparte de comercializar dispositivos de red al completo, vende también las RouterBOARD, que son circuitos integrados, preparados para ser montados en cualquier lugar.

MikroTik también dispone de sus propias certificaciones, en las que en toda su oferta incluyen seis. Uno de iniciación, y los otros cinco de especialización.

2.6.3. Juniper Networks

Juniper Networks es una multinacional dedicada a sistemas de redes y seguridad, fundada en 1996 por Pradeep Sindhu, con sede en California [30]. Sus productos incluyen routers, switches, software de control de redes, productos de seguridad de redes y tecnología SDN.

Figura 34: Logo de Juniper Networks

Juniper originalmente estuvo especializado en core routers, es decir, en routers diseñados para operar en la backbone de internet. Actualmente también diseñan routers para ISPs, y productos de seguridad. Y recientemente se han enfocado en el desarrollo de productos SDN (Software-Defined Networking).

2.7. XML

Sus siglas significan, en inglés, *eXtensible Markup Language*, o en español, "Lenguaje de Marcas Extensible", es un lenguaje de marcas desarrollado por el W3C (*World Wide Web Consortium*) que define una serie de reglas para codificar documentos, de forma que sea fácilmente leído por humanos, y también sencillo de parsear para las máquinas [31].

Figura 35: Logo de XML

El objetivo de diseño de XML fue la simplicidad, generabilidad y usabilidad para su uso en Internet. Es un formato de texto con mucho soporte vía Unicode. Aunque fue diseñado con el objetivo de formar documentos, está ampliamente usado para la representación de estructuras de datos arbitrarias.

Nació a partir del lenguaje de marcas SGML. De él también nació en un principio el lenguaje HTML [32]. Las principales diferencias que tiene con su hermano HTML es que las marcas (o *tags*) en HTML ya están definidas, mientras que en XML esas marcas no están predefinidas, y puedes definir tú mismo una. Esto es lo que hacen muchos fabricantes de routers, creando su propio "vocabulario" de marcas XML, creando así un nuevo *namespace*.

Namespaces

¿Qué son los *namespaces*? Es una cuestión que surgió al querer unificar ciertos documentos en que se usaban las mismas marcas para diferentes elementos. Para evitar eso, en cada marca, le precede un sufijo especificativo.

Cuando usamos esos prefijos en XML, un *namespace* debe definirse para el prefijo. Esto se debe definir por un atributo xlmns en la primera marca de un elemento, o también otra forma es declararlos como el *root* del XML.

Si declaramos un namespace con una URI (*Universal Resource Identifier*) nos ahorraremos entonces declarar un prefijo para cada elemento que nos encontremos.

XPath

No querríamos terminar sin conocer un poco más XPath, el lenguaje de búsqueda de información para documentos XML. XPath usa expresiones para seleccionar nodos, muy parecidas a las expresiones de rutas con las que se trabaja con un sistema operativo común.

Ejemplos de expresiones:

- */librería/libro[1]* : Selecciona el primer elemento libro que es hijo del elemento librería
- */librería/libro[position()<3]* : Selecciona los dos primeros libros que son hijos del elemento librería
- *//título[@lengua='es']* : Selecciona todos los elementos título que contienen un atributo llamado "lengua" con un valor de "es"

2.8. ¿Por qué se ha elegido la opción propuesta?

A lo largo del capítulo dos se ha realizado un recorrido por una serie de herramientas alternativas con las que trabajar con servidores TFTP y HTTP con DHCP Option 66. Las distintas tecnologías de ACS para el Technical Report 069, los diferentes lenguajes de programación para conectar con las distintas bases de datos y realizar una interfaz sencilla para el usuario. También hemos visto formas de trabajar con lenguajes de marcas para documentación, así como el estado actual de las bases de datos, importantes marcas de routers, etc. Por último, hemos mencionado el lenguaje XML. A continuación explicaremos por qué hemos elegido las opciones propuestas y/o por qué hemos necesitado hablar del estado del arte de estas tecnologías:

- Para el servidor TFTP, escogimos Xinetd, debido a que era mucho más sencillo de mantener, consumía menos recursos y se podría alojar en el mismo servidor en el cual estábamos desarrollando el código, en un Ubuntu Server 14.04.

- Para el servidor HTTP, escogimos Apache HTTP Server. Esto es debido a que su instalación es muy sencilla y no requeríamos ninguna concurrencia. La tasa a la que un nuevo router de voz se configura es muy baja, y por lo general nunca habrá concurrencia.

- Para el Auto-Configuration Server decidimos usar una opción Open Source, debido a que el presupuesto no alcanzaba para usar software privativo, ni interesaba debido al bajo volumen de dispositivos ADSL que íbamos a implantar. Decidimos escoger GenieACS, que está en auge y su moderna tecnología asegura que el proyecto seguirá mejorando y optimizándose durante muchos años.

- Para el lenguaje de programación, escogimos R por su facilidad de integración con todo lo que queríamos hacer. Aparte de que es más sencillo trabajar con datos y bases de datos (que era lo que fundamentalmente íbamos a hacer, y es con lo que mejor trabaja R), dispone del módulo Shiny preparado específicamente para desarrollar páginas web reactivas, así como un paquete de lenguaje de marcado RMarkdown. La optimización no era un requisito. Además, hemos usado Python para una parte del código cuando llama al

sistema de tickets de la empresa. Para recalcar la rapidez del programa aun usando R, la parte del código escrito en Python es la más costosa en tiempo.

- Para el lenguaje Markdown, hemos utilizado RMarkdown, ya que venía implementado en el mismo lenguaje en que estábamos progamando, y nos facilitaba mucho las tareas, además de que en todos los demás lenguajes el código Markdown es prácticamente el mismo.

- En cuanto a bases de datos, las que había en la empresa eran o MySQL o PostgreSQL, así que esas son las principales bases de datos a las que consultaremos desde R.

- Y para los routers, los que se encargan del routing en la empresa son de la marca MikroTik, que configuramos para el DHCP66, y los routers destinatarios para clientes eran Tenda y Grandstream.

- Por último, usaremos la información adquirida de los documentos XML para configurar los routers de voz Grandstream, usando el paquete XML de R.

Tras estas explicaciones, procederemos a explicar más a fondo y de forma más práctica la tecnología utilizada.

Capítulo 3. Tecnología utilizada.

Siguiendo con el índice descrito en el capítulo 1, se va a comenzar a exponer el tercer capítulo. En este apartado se va a desarrollar más profundamente todos los aspectos relativos a las herramientas utilizadas durante el trabajo.

Esta sección ayudará a comprender todos los aspectos teóricos de las plataformas, su funcionamiento y se realizarán ejemplos que ayudarán a entender mejor las tecnologías de forma práctica.

3.1. Xinetd

Como ya explicamos anteriormente, es mucho más que un servidor TFTP, y es conocido por ser un super-servidor, ya que gestiona los diferentes servidores de cada servicio. Por eso es que profundizaremos más acerca de los detalles técnicos y sus posibles usos en una máquina Linux.

Figura 36: Seguridad, ante todo, en xinetd

Xinetd brinda una excelente defensa contra intrusiones, y específicamente conveniente al reducir los riesgos de ataques *DoS* (Denial of Service). Permite también corregir los permisos de acceso a una máquina, si bien puede hacer mucho más que esto.

Los servicios definidos en los archivos de configuración pueden ser divididos en dos grupos. Los del primer grupo son llamados multihilo, y requieren la creación de un nuevo servidor de procesos para cada nueva petición de conexión. Tras eso, el nuevo servidor maneja esa conexión. Para esos servicios, xinetd sigue escuchando para crear nuevos servidores. Los del segundo grupo incluyen servicios para lo cual el demonio es responsable de manejar todas esas nuevas peticiones de conexión. Esos servicios se llaman monohilo, porque xinetd parará de escuchar nuevas peticiones hasta que el servidor termine.

La principal razón de por qué fue concebido como un super-servidor fue para evitar la creación masiva de procesos que seguramente estarían en reposo (Ya que siempre estarían activos incluso sin peticiones de conexión). Además de esto, se sirve de ser un super-servidor para poder ofrecer servicios de control de acceso y logueo. Asimismo, se vuelve especialmente interesante porque no está limitado a servicios de servidores, sino que cualquier persona podría usar xinetd para arrancar servidores de propósito específico.

Según la especificación, las opciones para lanzar xinetd son las siguientes [ref http://linux.die.net/man/8/xinetd]:

-d:

Habilita el modo de debugging.

-syslog *syslog_facility*:

Habilita el syslog usando la herramienta que le hemos puesto como opción, escogida entre los siguientes nombes: *daemon, auth, user, local[0-7]*.

-filelog *logfile*:

Los mensajes se enviarán al fichero elegido.

-f *config_file*:

Determina el archivo de configuración que xinetd cargará. Por defecto está en */etc/xinetd.conf.*

-pidfile *pid_file*:

La PID será escrita en el fichero.

-dontfork:

Esta opción hace que xinetd se oculte antes que permanecer activo, si no hay ningún servicio activo.

-stayalive:

Esta opción hace que xinetd persista incluso si no hay servicios especificados.

-limit *proc_limit*:

Da lugar a un límite de procesos concurrentes que puede lanzar xinetd contra otros servidores. Previene contra *overflows* de tablas.

-logprocs *limit*:

Da lugar a un límite de servidores concurrentes activos para adquisiciones remotas del *userid.*

-version:

Muestra información sobre la versión de xinetd.

-inetd_compat:

Esta opción hace que xinetd lea */etc/inetd.conf* en adición a los archivos de configuración estándar de xinetd.

-cc *interval*:

Hace que cada tiempo especificado como *interval* haga una revisión de consistencia de su estado interno.

Xinetd realiza ciertas acciones cuando recibe ciertas señales. Las acciones asociadas

con las señales específicas pueden ser redefinidas editando *config.h* y recompilando. Son las siguientes:

- **SIGHUP**. Provoca una profunda reconfiguración, que significa que vuelve a leer el archivo de configuración y termina los servidores cuyos servicios no están disponibles.
- **SIGQUIT**. Provoca la terminación del programa**.**
- **SIGTERM**. Termina todos los servidores activos antes de terminar con xinetd.
- **SIGUSR1**. Provoca un estado interno de volcado en */var/run/xinet.dump*.
- **SIGIOT**. Provoca una revisión de consistencia de su estado interno, para verificar que las estructuras de datos no han sido corruptas.

Ejemplo de servidor TFTP

Procederemos a hablar ahora acerca del archivo ubicado en */etc/xinetd.conf*. Por lo general este archivo tiene entradas de la forma:

```
service <service_name>
{
<attribute> <assign_op> <value> <value> ...
...
}
```

Los servicios disponibles podrían ser, entre otros, TFTP, FTP y TELNET. Como dijimos anteriormente, podemos definir cualquier tipo de servicio, por ejemplo, SMTP, HTTP o SSH, pero esos servicios se usan en modo *standalone* debido a que son servicios que contienen mucho tráfico, ya que xinetd los manejaría de un modo más lento.

El archivo de configuración que generamos, por ejemplo, es el siguiente:

c

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /tftpboot
disable         = no
}
```

Lo creamos en la ubicación */etc/xinetd.d/tftp*. A continuación explicaremos el significado de cada atributo.

- El atributo *protocol* sirve para establecer el tipo de comunicación, entre tcp o udp. En este caso, el protocolo tftp, por definición, trabaja en udp.

- En el atributo *port* especificamos el puerto en el que queremos que escuche. En

nuestro caso el 69, ya que es el puerto por definición del protocolo tftp.

- En el atributo *socket_type* especificamos el tipo de socket que manejará el servidor. Entre las opciones están *stream, dgram, raw* y *seqpacket*. Usaremos *dgram* ya que será un servicio basado en datagramas.

- El atributo *wait* determina si el servicio es monohilo o multihilo. Si el valor es *yes*, el servicio será monohilo, que quiere decir que xinetd arrancará el servidor y parará de escuchar nuevas peticiones hasta que el servidor muera. Según la recomendación del desarrollador, para configuraciones *udp/dgram* se espera que este atributo sea *yes*, ya que udp no está orientado a conexión.

- El atributo *user* determina el *uid* para el proceso del servidor. Si se da un nombre, ese usuario debe existir en */etc/passwd*.

- El atributo *server* determina el programa que ejecutar para este servicio.

- El atributo *server_args* son los argumentos que se pasarán al programa que se ejecutará. En nuestro caso, para el servidor tftp, será el directorio en donde nuestro servidor tftp buscará archivos que pidan.

- El atributo *disable* determina si el servicio se deshabilitará o no.

Con esto, ya tendremos nuestro servidor casi listo. Sólo nos falta crear y modificar los permisos para hacer funcional el directorio */tftpboot*.

```
sudo mkdir /tftpboot
sudo chmod -R 777 /tftpboot
sudo chown -R nobody /tftpboot
```

Donde */tftpboot* debe coincidir con el directorio que le hayamos pasado como argumentos al programa en el fichero de configuración de xinetd, y lo mismo sucede para el usuario *nobody*.

Tras esto, reiniciaremos el servicio xinetd.

```
sudo service xinetd restart
```

Y tras esto, nuestro servidor ya estará completamente listo para funcionar. Vamos a probarlo.

Primero crearemos un archivo de test para comprobar su funcionalidad, llamado *test*, que guardaremos en la carpeta */tftpboot*.

Figura 37: Creación del archivo de test

Una vez creado, lo que haremos será ejecutar un cliente tftp. Para nuestra prueba, usaremos el cliente para Linux llamado *tftp*. Para conectarnos a nuestro servidor tftp, lo que haremos será ejecutar tftp seguido de la dirección, para a continuación extraer el archivo.

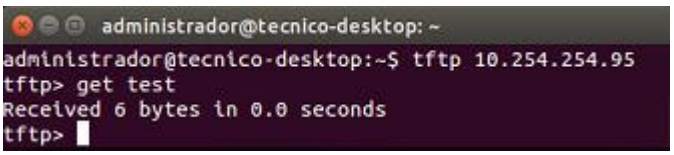

Figura 38: Petición de archivo de test vía tftp

Y observamos que se ha transferido correctamente:

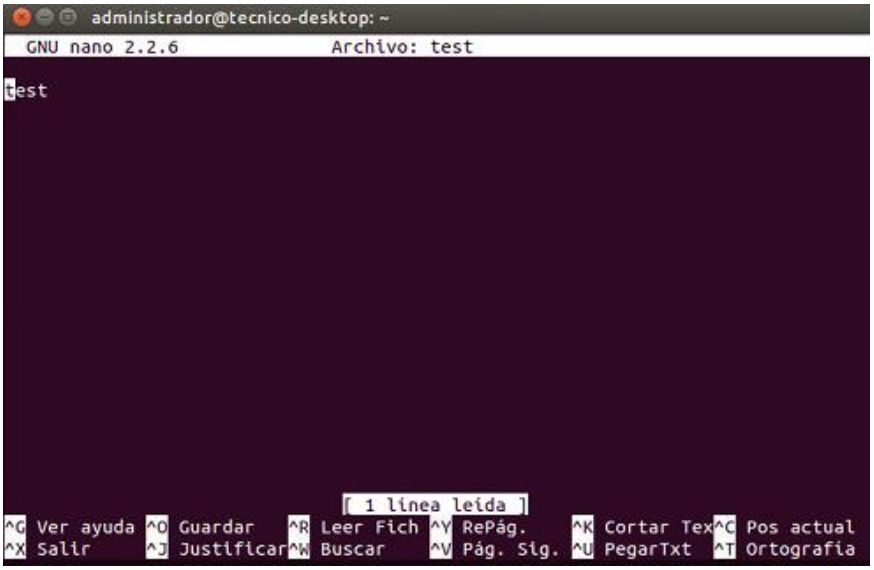

Figura 39: Comprobación de archivo recibido

3.2. Apache HTTP Server

Como ya hemos explicado antes, es uno de los servidores web más veteranos de todo internet, por ello la documentación abunda muchísimo, y es muy sencillo de instalar.

Procederemos a exponer un ejemplo de instalación cualquiera, bajo un entorno Linux.

El único comando que necesitar para instalarse es:

```
sudo apt-get install apache2
```

Con la que ya tendremos nuestro servidor funcionando. Pero para hacer lo que nosotros queremos, tenemos algunos archivos de configuración que detallamos a continuación:

- *Apache2.conf:* Configuraciones globales para Apache2
- *envvars*: Variables de entorno establecidas
- *ports.conf:* Directivas que determinan los puertos TCP donde apache escuchará

Entre otros. Así de sencillo.

Ahora bien, ¿cómo creamos esos archivos de configuración que necesitan los routers?

3.3. El entorno de trabajo RStudio, R, y el paquete Shiny

Comenzaremos profundizando el IDE RStudio y con los elementos de programación debido a que, para explicar el resto, debemos saber cómo funciona R para saber cómo podemos implementar la creación de los archivos de configuración y las sincronizaciones con las demás bases de datos. Posteriormente también hablaremos de Python, ya que también lo hemos usado.

3.3.1. RStudio

RStudio está disponible en su versión de escritorio para Windows, iOS, Debian/Ubuntu y Fedora/RedHat/openSUSE [33]. En su versión servidor para Debian/Ubuntu, RedHat/CentOS y openSUSE/SLES, aunque virtualmente se podría hacer en cualquier SO debido a que tenemos disponible la descarga del código fuente, para instalar desde él.

Una vez instalado (en el Anexo A se hablará de cómo instalarlo en su versión servidor), la aparencia es parecida a esta en las diversas plataformas:

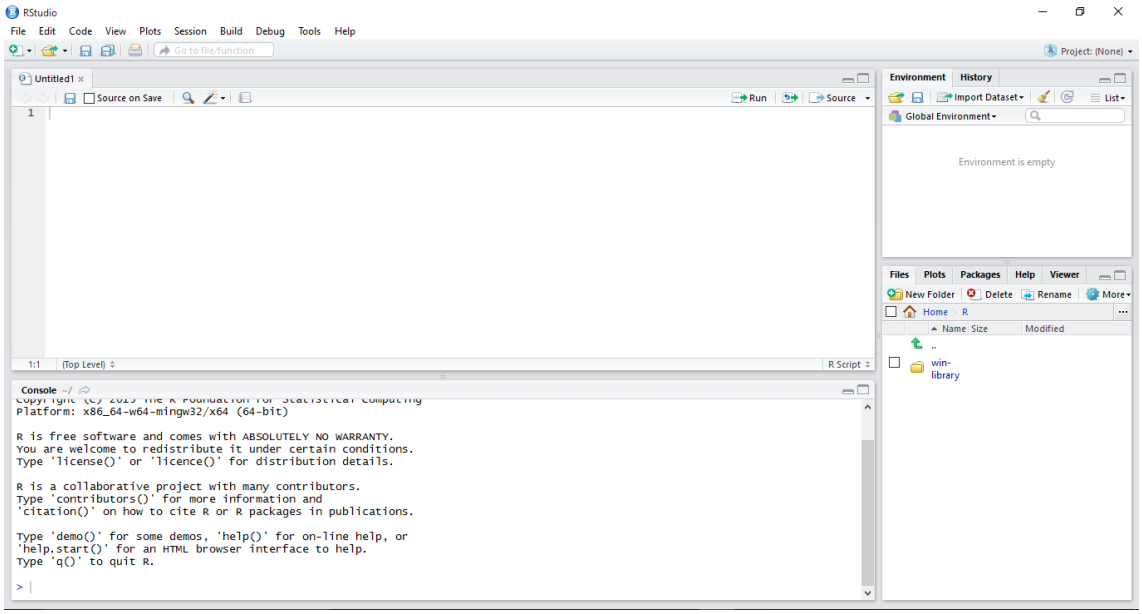

Figura 40: Entorno de trabajo RStudio

En la versión servidor, la GUI sería la misma, sólo que dentro de un navegador, al que accederíamos a través del puerto 8787 vía HTTP.

En la imagen podemos ver que tenemos una barra superior de herramientas, donde podremos abrir, cerrar ficheros, editar código por varios métodos de búsqueda y reemplazo… Muchas opciones, y nos gustaría destacar una.

En *Tools*, iremos a *Global Options*. Desde allí encontramos todas las opciones de configuración para poner a punto nuestro IDE.

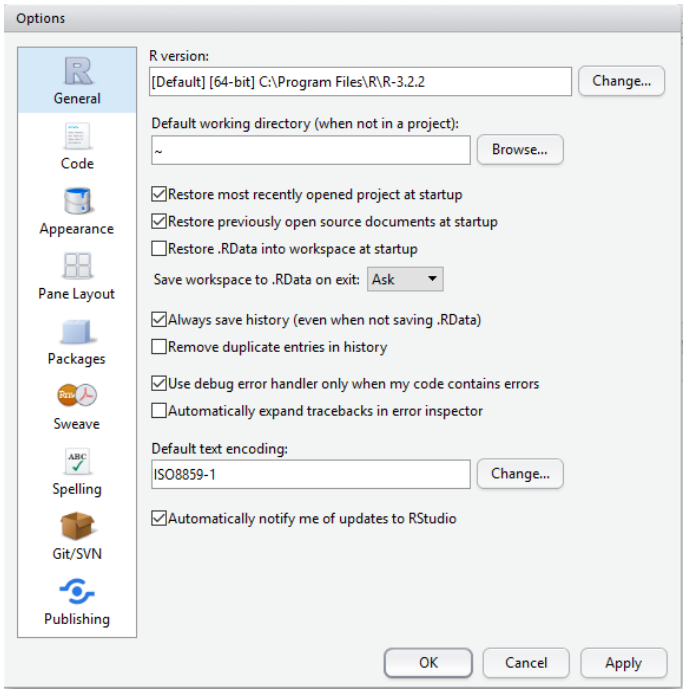

Figura 41: Configuraciones Globales de RStudio

Aquí podremos elegir una versión diferente de R que la instalada, o escoger el directorio de trabajo por defecto. Después tenemos una serie de opciones, como guardar el espacio de trabajo al cerrar. Más abajo tenemos otra opción muy interesante, y es la elección de codificación por defecto. Podremos escoger entre varias: ISO, UTF, ASCI… Una elección bastante importante según con qué datos estemos trabajando y en qué sistema operativo.

Volviendo a la pantalla inicial, encontramos en la parte izquierda superior, un editor de texto, donde escribiremos nuestro código. Más abajo, en la parte izquierda inferior, encontramos la zona donde ejecutar nuestro código. Allí encontraremos los logs, resultados, errores y warnings.

Arriba a la derecha nos encontramos, en la parte superior, con una zona donde nos encontramos el entorno de trabajo, donde se pueden ver todas las asignaciones de variables que hay en este momento. Podemos ver su tipo e incluso lo que ocupa en memoria. También podemos ver que contiene otra ventana para seleccionar, donde encontramos el historial de comandos que hemos introducido en la consola.

En la parte inferior, a la derecha, encontraremos varias funcionalidades. Podemos ver un listado con los directorios, partiendo desde el directorio de trabajo. Además, podemos ver los *plots* resultantes de nuestras operaciones. También podemos en la siguiente pestaña, trabajar con los paquetes de R, instalándolos, desinstalándalos o cargándolos en memoria. En la siguiente pestaña tenemos una sección de ayuda para cuando ejecutamos comandos de solicitud de ayuda para ciertas funciones. La última pestaña se utiliza para ver contenido web en local, por ejemplo, con Shiny.

Ahora que hemos hecho un repaso al IDE, vamos a profundizar más acerca de R:

3.3.2. R

Como ya hemos explicado antes, R es un lenguaje especialmente enfocado al análisis de datos, siendo muy sencillo trabajar con información. Hablemos de su paradigma.

R soporta una mezcla de orientación a objetos con programación funcional en los siguientes sentidos:

Por parte de programación funcional, tenemos:

- Funciones de primera clase.
- Evaluación perezosa de argumentos (*lazy evaluation*).
- Funciones puras, sin efectos secundarios.

Sin embargo:

- No implementa colas de llamadas de optimización (*Tail call recursion*).
- Es sencillo crear funciones impuras.

Por parte de orientación a objetos tenemos:

- Tres paradigmas de orientación a objetos: S3 y S4, que son inmutables y dan

soporte a funciones genéricas, y R5, o clases de referencia, que son mutables, pues dan soporte a traspasos de mensaje más comunes.
- El paradigma S4 está muy influenciado por common lisp (CLOS) y dylan.
- Existen paquetes que proveen otros tipos de paradigma de orientación a objetos, como *proto, mutatr, R.oo* y *OOP.*

Sin embargo:
- Las herramientas disponibles en R para orientación a objetos proveen poco desarrollo en su facilidad para implementarlos (también conocido en inglés como *syntactic sugar*)

El lenguaje soporta *lexical scoping*, que explicaremos de la siguiente forma:

Consideremos la siguiente función en R:

```
f <- function(x, y) {
    x^2 + y / z
}
```

Esta función tiene dos argumentos formales, x e y. Pero nos damos cuenta que en el cuerpo de la función, hay otro símbolo llamado z. En este caso, z se le conoce como una variable libre. Las reglas de *scoping* de un lenguaje definen qué valores pueden ser asignados a variables libres.

El *lexical scoping* en R significa que los valores de las variables libres son buscadas en el entorno en que la función fue llamada.

¿Qué es un entorno? Es una colección de pares símbolo-valor. Por ejemplo, x es un símbolo y 3.14 puede ser su valor. También es sabido que un entorno puede tener padres e hijos. El único entorno sin padre es el entorno vacío.

Así que cada vez que R debe buscar el valor de esa variable, la busca en el entorno donde la función se definió. Luego sigue hacia arriba de padre en padre hasta que llega al entorno vacío. Si aún no se ha encontrado el valor, se lanza un error.

Figura 42: Logo de la conferencia internacional anual sobre R

Hablaremos ahora de las funciones de entrada y salida de datos, que tendremos que usar para leer los ficheros y escribirlos a disco, así como las herramientas que se pueden usar para la modificación de esos archivos.

Input/Output de datos

Para leer datos de disco, necesitaremos saber el tipo de datos con el que trabajaremos. Podríamos leerlos todos con un mismo método, pero tardaríamos un costoso tiempo en transformar al formato de presentación que queramos en R.

Para leer archivos delimitados por comas (comúnmente, CSVs), tenemos la función *read.table()*:

```
mydata <- read.table("c:/mydata.csv",
        header=TRUE, sep=",", row.names="id")
```

Donde tendremos que poner los argumentos que correspondan en cada caso. En este ejemplo hemos supuesto que el archivo se inicia con un *header*, del que extraemos los nombres de cada columna,

También tenemos una forma más directa de hacerlo, con la función *read.csv()*:

```
mydata <- read.csv("c:/mydata.csv")
```

Que en realidad es la misma función que la que hemos mencionado antes, sólo que esos parámetros en esta función están por defecto.

Recientemente también ha habido progresos en la forma de leer datos, y para cantidades grandes de datos, la función *fread()* del paquete *data.table* se destaca por ser el más rápido de todos los lectores y el lector con mayores funcionalidades [34]. Es de destacar que la versión todavía está en desarrollo. La sintaxis es diferente y compleja, y procederemos a explicar con más detalle esta función:

```
fread(input, sep="auto", sep2="auto", nrows=-1L, header="auto",
        na.strings="NA", stringsAsFactors=FALSE, verbose=FALSE,
        autostart=30L, skip=-1L, select=NULL, colClasses=NULL,
        integer64=getOption("datatable.integer64"))
```

Argumentos:

input

El archivo que leer, o un mismo string que deseemos darle formato. También puede ser una URL

sep

El separador que exista entre las columnas. Si se especifica en “auto”, se establece como el primer valor que se encuentra en este set [,\t |;:] a partir de la línea especificada en el argumento *autostart*.

sep2

El separador interno de cada columna. Si existe, se devolverá una lista en vez de un data frame. Este es mucho más rápido que invocar la función *strsplit* tras leer

el data frame. Al igual que el argumento sep, si el modo "auto" está activado, se establece como valor el primero encontrado en este set [,\t |;:].

nrows

Número de filas que leer. Por defecto, -1 significa todas. No ayuda a ser más veloz el poner como argumento el número de filas real del archivo.

header

Si la primera línea del fichero contiene los nombres de las columnas, este argumento las obtiene.

na.strings

El símbolo que definamos para un valor NA (Non-Available). Por defecto es el string "NA".

stringsAsFactors

Si está activado, convierte todas las columnas carácter en factores.

verbose

Activar del modo verboso

autostart

Cualquier línea dentro de la región de texto delimitado por la capacidad de lectura de la máquina, que por defecto se establece a 30. Si el fichero es más corto, o la línea está vacía, entonces la última línea con texto es utilizada. Estas líneas y las líneas superiores serán las que sep y sep2 utilizarán para la autodetección.

skip

Si se establece a -1, se usa el procedimiento descrito más arriba, empezando por la línea definida por el argumento autostart, para encontrar la primera fila de datos. Si se establece a 0 o mayor que 1, significará que el argumento autostart se ignorará, y cogerá la línea skip+1 como la primera fila de datos.

select

Aún no implementado. Vector de columnas o posiciones que guardar.

colClasses

Establece un vector de caracteres con las clases, que le pasemos como argumento, tal como en la función *read.csv()*.

integer64

Aún no implementado. Lee columnas detectadas como enteros mayores que 2^31 como tipo bit64::integer64.

Ahora bien, ¿cómo guardamos los archivos a disco? Hay varias opciones a considerar.

Si queremos velocidad, y el archivo sólo sirve para extraer datos, entonces podemos usar la función *save()*. Esta función guarda a disco como objeto R los objetos que le pasemos como parámetro. Este tendrá la extensión .RData .

Si necesitamos conservar el formato del archivo, haremos uso de diversas funciones disponibles, como *write(),* que es simplemente un *wrapper* de la función *cat()*. También disponemos de las funciones *write.table()*, y *write.csv()*, que sirven para escribir data frames a disco.

Para leer datos desde un formato Microsoft Excel, usaremos el paquete *xlsx*.

```
library(xlsx)
mydata <- read.xlsx("c:/myexcel.xlsx", 1)
mydata <- read.xlsx("c:/myexcel.xlsx", sheetName = "mysheet")
```

En este ejemplo hemos cargado en el espacio de trabajo el paquete *xlsx* mediante la función *library()*. Para que todo marche bien, deberemos tener instalado el paquete *xlsx*, que se instala por medio de la función *install.packages()*.

Si queremos guardar a disco el fichero Excel, usaremos la función *write.xlsx()*.

Si lo que queremos es abrir archivos con formato XML, deberemos de usar el paquete *XML* de R. En él se hallan muchas formas de leer y parsear archivos XML, así como muchísimas funciones para trabajar con ellos desde dentro de R.

Pongamos un ejemplo de cómo podríamos leer un archivo cualquiera XML, y trabajar con él.

```
library(XML)
archivo <- xmlParse("C:\\Users\\sergi\\Desktop\\archivo.xml")
posicion <- xpathSApply(archivo, "//P2")
```

Esta parte del script primero carga en memoria el paquete XML, para luego usarlo invocando a la función *xmlParse()*. Hay más funciones para parsear archivos XML, así como XHTML, como las funciones *htmlParse()*, *htmlTreeParse()*, o incluso la función *htmlParseDoc()* en la que puedes controlar muchas opciones de bajo nivel del parser.

En el elemento *archivo* podremos encontrar una represenación en forma de objeto en R. Si lo que queremos es buscar una marca que por ejemplo, se llame P2, invocaremos a la función *xpathSApply(),* usando notación de XPath.

Así en el objeto *posición* ya tenemos los resultados de la búsqueda por parte de XPath. Éste nos devuelve no la copia de un objeto, sino el mismo objeto, con una dirección apuntando a la localización del elemento que hayamos buscado.

Es por eso que podemos hacer lo siguiente:

```
xmlValue(posicion[[1]]) <- "Valor asignado"
xmlChildren(posicion[[1]]) <- " - Extensión"
xmlName(posicion[[1]]) <- "Tag1"
```

Dejando el nodo XML de esta forma:

```
<Tag1>Valor asignado - Extensión</Tag1>
```

Así vemos que, primeramente, hemos cambiado el contenido del nodo XML al valor "Valor asignado". Tras esto, hemos añadido contenido en vez de borrar lo que había escrito, y hemos añadido el valor " - Extensión". Por último, hemos modificado el nombre de la marca y ha pasado de llamarse "P2" a "Tag1".

Por último, explicaremos también más detalladamente la función *saveXML()* del paquete:

```
saveXML(archivo,
        file="C:\\Users\\sergi\\Desktop\\archivo.xml",
        prefix = '')
```

En este ejemplo, hemos guardado nuestro archivo que hemos modificado en la misma localización de donde lo extrajimos. Es de notar que el argumento *prefix* lo debemos asignar a un string vacío, ya que de otro modo vuelve a escribir el inicio de fichero XML, que ya viene por defecto en nuestro archivo.

Conexiones a Bases de Datos

Parte fundamental de nuestro programa son las llamadas a las bases de datos, así que en esta sección explicaremos las formas de trabajar con ellas.

Depende de la base de datos a la que accedamos, necesitaremos un paquete diferente. En nuestro caso hemos conectado a bases de datos MySQL y PostgreSQL, así que explicaremos los paquetes asociados a estas bases de datos.

Para conectar con bases de datos MySQL, tenemos el paquete RMySQL. Un ejemplo de uso es el siguiente:

```
Library(RMySQL)
con <- dbConnect(MySQL(), user="usuario", password="contraseña",
        "dbname"="nombreDB", host="localhost", port= "3307")
query <- dbGetQuery(con, "SELECT * FROM libreria")
dbDisconnect(con)
```

Primero, precargamos el paquete RMySQL, tras lo cual creamos una conexión con la base de datos MySQL, usando la función *dbConnect()*. Insertamos argumentos como el usuario para la base de datos, la contraseña, la base de datos a la que queramos acceder, y la localización IP de la máquina. Incluso si el servidor MySQL no escucha por

el puerto por defecto, podemos cambiar el puerto.

Tras haber conectado a la base de datos de forma satisfactoria, hemos procedido a invocar a la función *dbGetQuery()*. Como argumentos, toma la conexión creada anteriormente, y el *query* que le queramos enviar a la base de datos, con un formato válido y adecuado a la estructura de *queries* para la base de datos MySQL.

En query tendremos nuestra respuesta, debidamente ordenada en un data frame. Por útimo no debemos olvidar cortar la conexión con la base de datos, ya que consume un espacio tanto en la máquina donde ejecutemos el código como en el servidor MySQL.

Si sólo quisiéramos enviar datos a la base de datos, usaríamos el comando *dbSendQuery()* en vez de *dbGetQuery()*, ya que éste último es más lento, pues lo que hace también internamente es formatear el resultado del retorno de datos.

Para conectar R a la base de datos PostgreSQL tenemos también el paquete *RPostgreSQL*. El funcionamiento es muy parecido al de MySQL:

```
Library(RPostgreSQL)
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, user="usuario", password="contraseña",
       "dbname"="nombreDB", host="localhost", port= "3307")
query <- dbGetQuery(con, "SELECT * FROM libreria")
dbDisconnect(con)
```

La única diferencia apreciable que nos encontramos es la de la declaración del elemento a conectar, invocando para ello a *dbDriver()*.

3.3.3. Shiny

Recientemente añadido a la lista de paquetes disponibles, es uno de los paquetes más útiles que nos podemos encontrar en CRAN [35]. Shiny es un entorno de trabajo para hacer aplicaciones web a partir de R, sin necesidad, en principio, de haber aprendido HTML, CSS o JavaScript, ya que Shiny hace todo esto por ti.

Figura 43: Ilustración artística de Shiny

Shiny es sencillo de aprender y hace gala de páginas web reactivas dependiendo de lo

que el usuario introduzca.

La aplicación consta simplemente de dos archivos: Uno de presentación, o de GUI, y otro dedicado al servidor (pudiendo incluso fusionarse en un único archivo). Un ejemplo de página sería este que presentamos a continuación [36].

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Hello World!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", min = 5,
        max = 50, value = 30)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

```
library(shiny)
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'skyblue', border = 'white')
  })
})
```

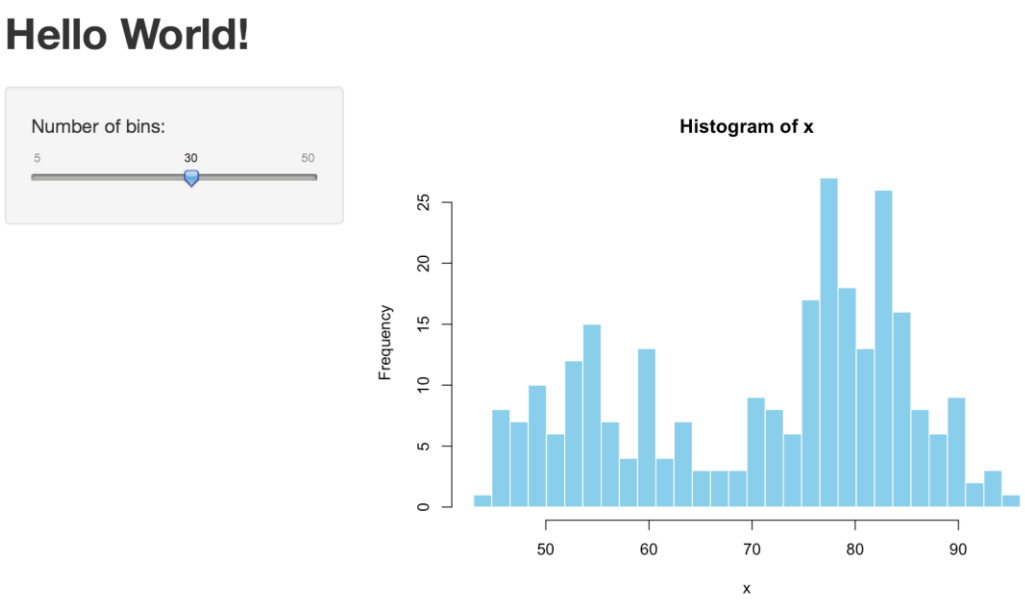

Figura 44: Ejemplo de app en Shiny

El primer bloque es la GUI de la aplicación, mientras que el segundo es la parte del

servidor. Procedamos a explicar cada una de las partes.

El script de GUI se empieza declarando la importación del paquete shiny. Tras esto, definimos la UI dentro siempre de un *shinyUI()*, que a su vez contiene un *fluidPage()*. Este *fluidPage()* se puede intercambiar por un *navbarPage()* si queremos que nuestra app tenga múltiples páginas conectadas por una barra de navegación. Si ponemos un *fluidPage()* simplemente tendremos una ventana donde se mostrará todo lo que pongamos, como vemos en la imagen.

Tras esto, vemos un *titlePanel()* que hará de título de la página web y de un encabezado de tamaño grande. Lo siguiente que nos encontramos es un *sidebarLayout()*, que define que la UI de la aplicación tendrá dos partes diferenciadas: un panel principal donde elegiremos los datos, y otro panel en el que se verán los resultados. Podemos elegir también entre un *splitLayout()*, que creará contenedores a partir de los inputs que le pongamos; o un *flowLayout()* que es parecido al *splitLayout()* sólo que los elementos no ocupan toda la pantalla y forman entidades movibles por toda la pantalla. Por último también disponemos de *fluidRow()* que define el espacio de UI como filas, en las que en cada fila pondremos todo lo que insertemos, y así hasta que no definamos la siguiente *fluidRow().*

Tras esto, nos encontramos con un *sidebarPanel()* que a su vez contendrá un widget, llamado *sliderInput*(), con el que nos dará como resultado un deslizador para elegir el número de particiones. Shiny aún está evolucionando en forma beta y cada día hay nuevos widgets que podemos elegir para nuestras aplicaciones. Están hechos en HTML, CSS y JavaScript.

Siguiendo, vemos un *mainPanel()* cuyo contenido es un *plotOutput()* simplemente con una referencia. Esto es una señal de que está llamando al servidor para saber qué plot debe insertar dentro. Esto forma parte de la dinamicidad de Shiny.

Comenzando a comentar el fichero del servidor, vemos que también definimos la importación del paquete Shiny, y tras esto siempre nos encontraremos con un *shinyServer()*.

Lo siguiente que nos encontramos es un *renderPlot({}).* Este *renderPlot({})* hará que la expresión que incluya dentro se ejecute de nuevo cada vez que haya un cambio en la aplicación. Forma parte de las expresiones reactivas de Shiny, tal como *renderUI({}), renderText({}), renderDataTable({})…* Pero sabemos que esta siempre devolverá un objeto tipo Plot.

Dentro vemos definido lo que inserta como plot en la UI, pero fijémonos en la asignación que se hace de *renderPlot({})* a *output$disPlot*. Esto quiere decir que llama a la referencia distPlot de la UI, para poder insertar allí el plot.

3.4. DHCP Option 66

Aunque ya hemos explicado cómo funciona DHCP66 de forma teórica, queríamos, en esta sección, explicar con detalle el proceso poniendo un ejemplo muy parecido al que implementamos finalmente [37].

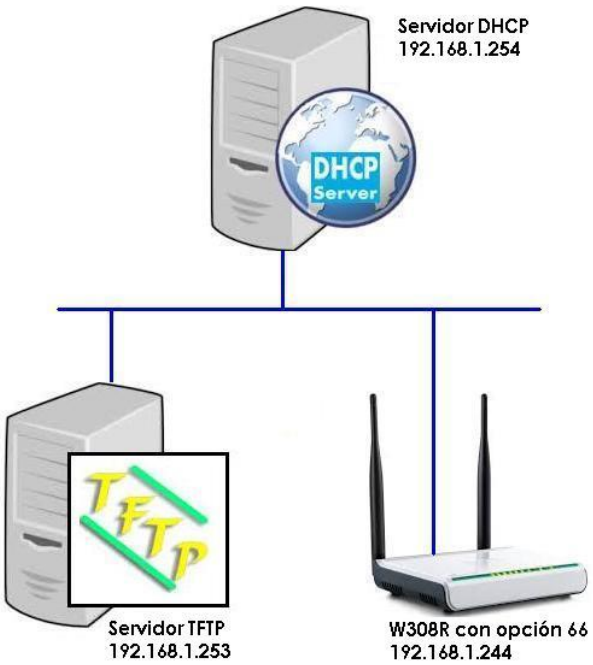

Figura 45: Esquema de red para DHCP66

En este ejemplo hemos usado un Tenda W308R, que parece ser uno de los pocos routers cliente en el mercado que acepta DHCP66.

Al conectar el puerto WAN en este router, si está en estado de fábrica, tras el paso inicial previo, enviará una petición con la opción 66 al servidor DHCP, que le dará la IP del servidor que hayamos configurado como servidor TFTP.

Al obtener nuestro router la IP del servidor TFTP, procedería a efectuar el comando GET para obtener el archivo asociado a su MAC, variando el nombre del archivo dependiendo del fabricante.

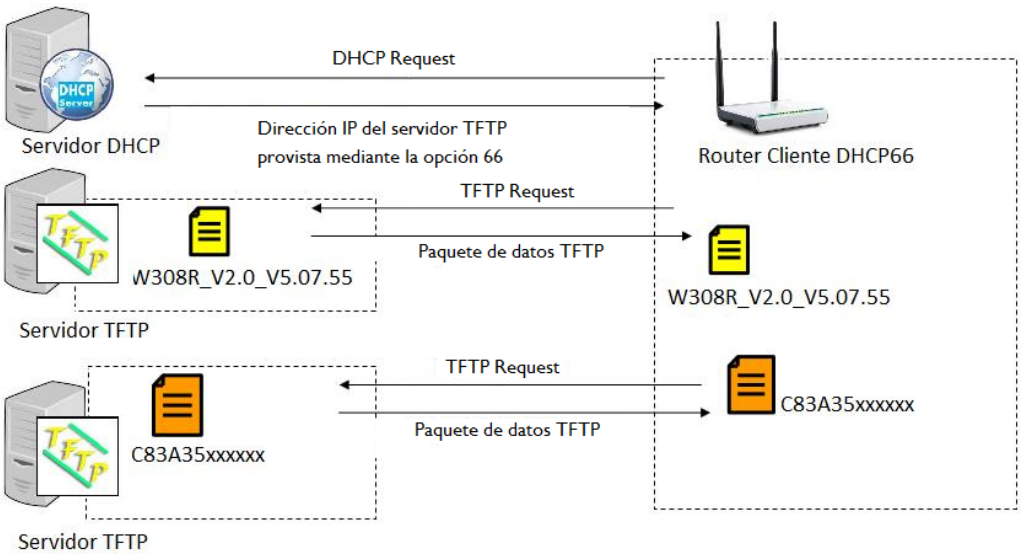

Figura 46: Ejemplo de consulta DHCP66

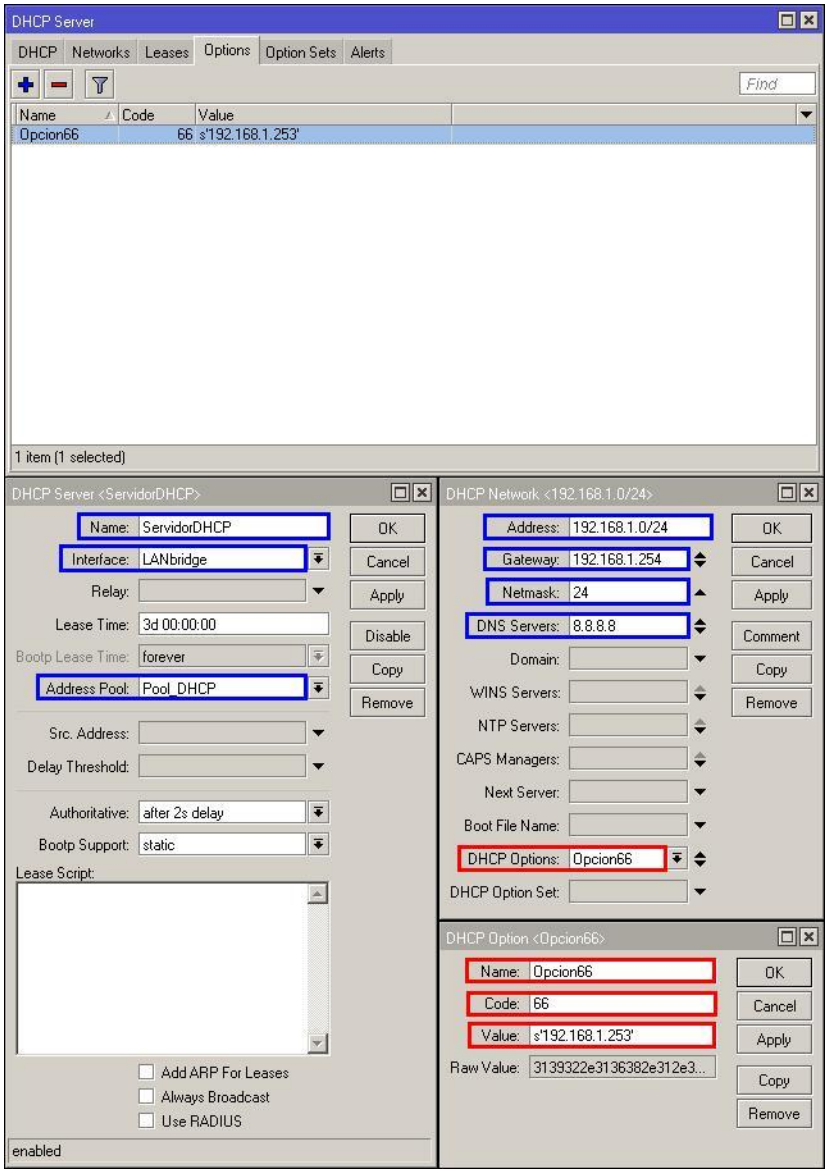

Figura 47: Configuración de DHCP Server Mikrotik, incluyendo DHCP66

Así que vemos, que de esta manera tan sencilla conseguimos que los routers cliente se autoconfiguren, sin necesidad de una gran inversión en equipamiento nuevo.

3.5. GenieACS

Nos hemos decantado por este software para el apartado de TR069 por su desarrollo, por su comunidad y por la total personalización que se le puede hacer. Así que vamos a demostrar de lo que es capaz esta pieza de software, y veremos cómo funciona su API. Finalmente veremos la configuración que podemos establecer desde los archivos de configuración.

Ejemplo de funcionamiento

Vamos a simular la entrada de un nuevo router a nuestro sistema. Se pueden establecer una serie de presets, con mayor o menor peso, para configurar este recién llegado CPE.

GenieACS da la opción de establecer un preset inicial, para uno o varios CPE, por GUI o por API. Primer explicaremos cómo se hace vía GUI.

Dentro de la pestaña de presets, nos encontramos con una pantalla donde podremos añadir un nuevo preset. Pondremos nombre, peso, precondiciones y configuraciones a efectuar, tal como vemos en esta imagen de ejemplo

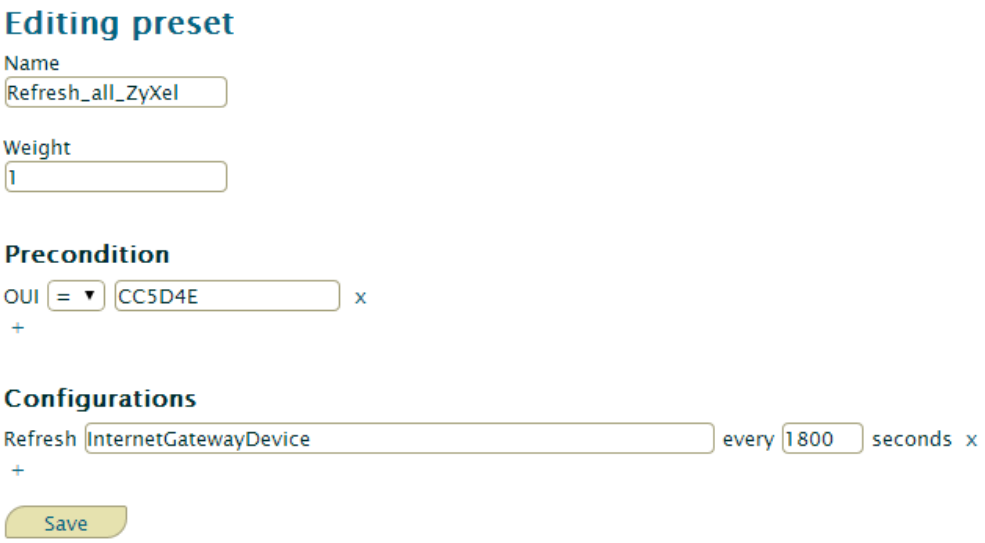

Figura 48: Ejemplo de creación de Preset mediante GUI

El nombre nos servirá para identificar el nombre del preset de un vistazo. El peso nos indicará el valor de ese preset, es decir, cuál se ejecutará antes si dos preset entran en conflicto de precondiciones. El que tenga el mayor valor entre ambos presets conflictivos, ese será el preset elegido para efectuar.

Entre precondiciones podemos elegir en versión del hardware del equipo, del software, del número serial, de la MAC… Y en la configuración podremos cambiar todos los presets que sean modificables.

Podemos automatizar la creación de presets mediante la API de GenieACS, que pasaremos a detallar a continuación [38]:

Para obtener acceso a la API, debemos haber activado el componente *genieacs-nbi* en nuestro servidor, que por defecto escucha en el puerto 7557. Usa HTTP y JSON como formato de datos.
Lo único que tendríamos que hacer sería hacer un GET desde un simple navegador para que se ejecutara lo que le mandásemos.

Pondremos el siguiente ejemplo para entenderlo mejor: Crearemos un preset para que todos los CPE marcados con el tag “test”, informe cada 5 minutos de su estado.

El preset en JSON sería el siguiente, para un CPE en general:

```
query={
   weight: 0,
   precondition: "{\"_tags\":\"test\"}"
   configurations:
      [
          { type: "value",
            name: "InternetGatewayDevice.ManagementServer.PeriodicInformEnable",
           value: "true" },
          { type: "value",
             name: "InternetGatewayDevice.ManagementServer.PeriodicInformInterval",
           value: "300" }
      ]
}
```

Por lo que, en base a este preset en JSON, deberemos de formatearlo para introducirlo en una cadena de texto como parte de la URL a la cual tendremos que acceder.

```
curl -i 'http://localhost:7557/presets/inform' \
-X PUT \
--data '{ "weight": 0, "precondition": "{\"_tags\":\"test\"}", "configurations": [
        { "type": "value", "name":
        "InternetGatewayDevice.ManagementServer.PeriodicInformEnable", "value":
        "true" }, { "type": "value", "name":
        "InternetGatewayDevice.ManagementServer.PeriodicInformInterval", "value":
        "300" } ] }'
```

Esto solo ha sido un ejemplo de lo que podemos hacer vía API, pero en realidad podemos enviar todo tipo de acciones. Desde devolver datos de un CPE en específico, pasando por poder borrar tareas especificadas para ciertos CPE, hasta hacer que un CPE se reinicie.

3.6. RMarkdown

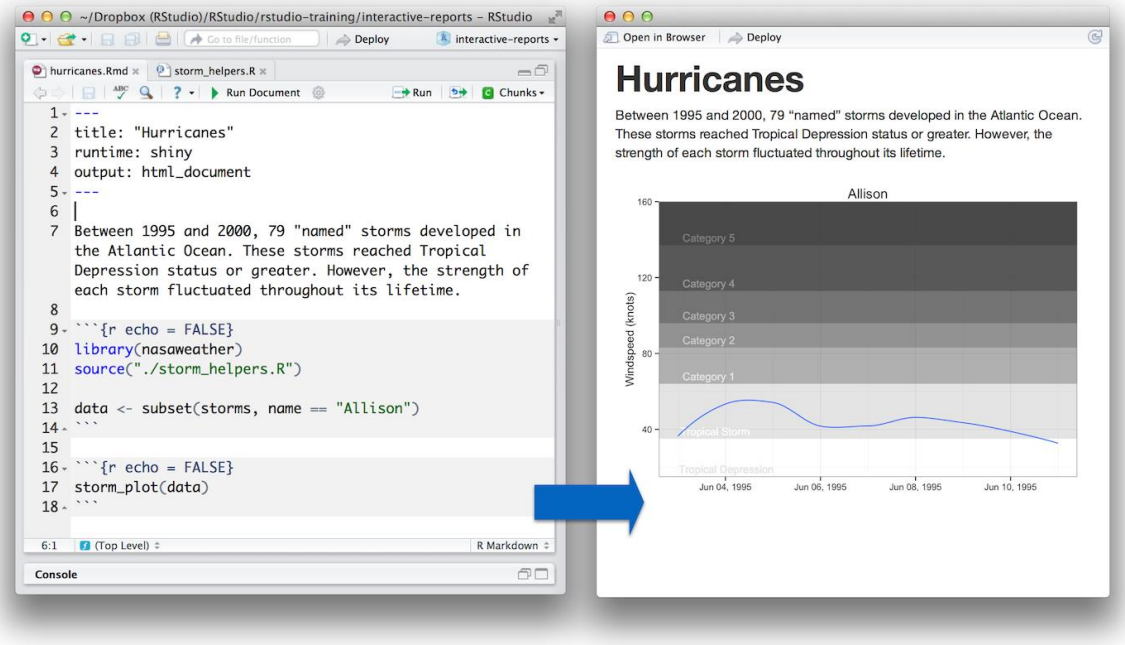

Figura 49: Ejemplo de documento RMarkdown y su output en HTML.

Es el editor de Markdown por defecto en R. Combina la sintaxis de markdown y bloques de código R que son ejecutados para ser visualizados en el documento final (generalmente, imágenes) [39].

Entre sus novedades más recientes, podemos encontrar:

- Muchos formatos de salida, incluyendo HTML, PDF y MS Word
- Soporte para la creación de presentaciones Beamer, ioslides y Slidy
- Soporte extendido para tablas y bibliografías
- Facilidades para customizar las salidas HTML y PDF, incluyendo CSS
- Incluye LaTeX en su estado puro para customización avanzada de ficheros PDF
- Compila notas de texto HTML, PDF o MS Word desde scripts R
- Extensibilidad: Crea plantillas customizadas e incluso nuevos formatos de salida
- Posibilidad de crear documentos interactivos usando Shiny

La sintaxis es muy sencilla, y es la que podemos ver en la foto. Las partes coloreadas son código R que es ejecutado y compilado para formar la imagen que vemos a la derecha.

Capítulo 4. Router Express

Habiendo explicado toda la tecnología envuelta en el desarrollo de este servicio, procederemos a explicar las funcionalidades y la implementación de cada parte de este servicio Web, así como de la configuración de los servicios asociados de DHCP Option 66 y Technical Report 069

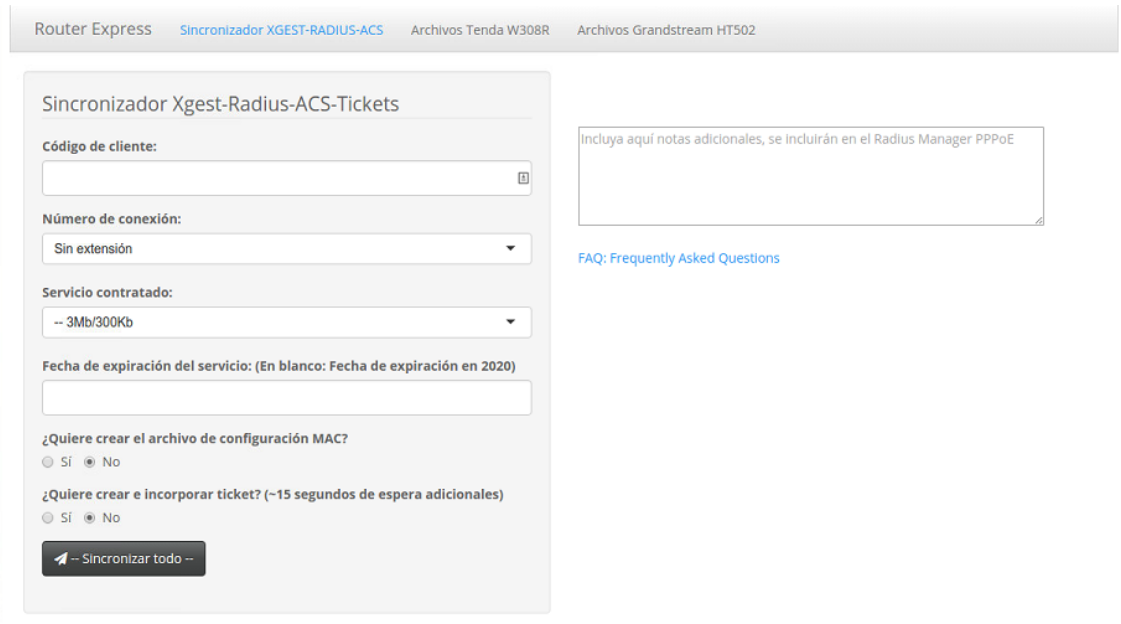

Figura 50: Vista principal de la WebApp basada en el paquete Shiny

Todos los servicios están alojados en una misma máquina virtual dentro del servicio de virtualización Hyper-V de Microsoft. La máquina virtual que hemos utilizado es un Ubuntu 14.04 LTS Server. Le hemos asignado 2 núcleos y 1,5GB de RAM.

4.1. Explicación de uso

Usaremos esta aplicación tras haber incorporado al primer cliente en la base de datos de nuevos clientes en la empresa. Estando ya el nuevo cliente creado, procederemos a usar esta aplicación para:

- Insertar usuario PPPoE al servidor RADIUS de clientes.
- Insertar usuario HotSpot al servidor RADIUS de HotSpot.
- Insertar usuario en la base de datos de tickets.
- Crear ticket de instalación para el nuevo cliente.
- Crear archivo de configuración para el Tenda W308R.
- Crear archivo de configuración para el Grandstream HT502.
- Crear un preset en GenieACS para el Tenda W300D.

Procedamos a explicar un ejemplo de uso, el más completo de todos.

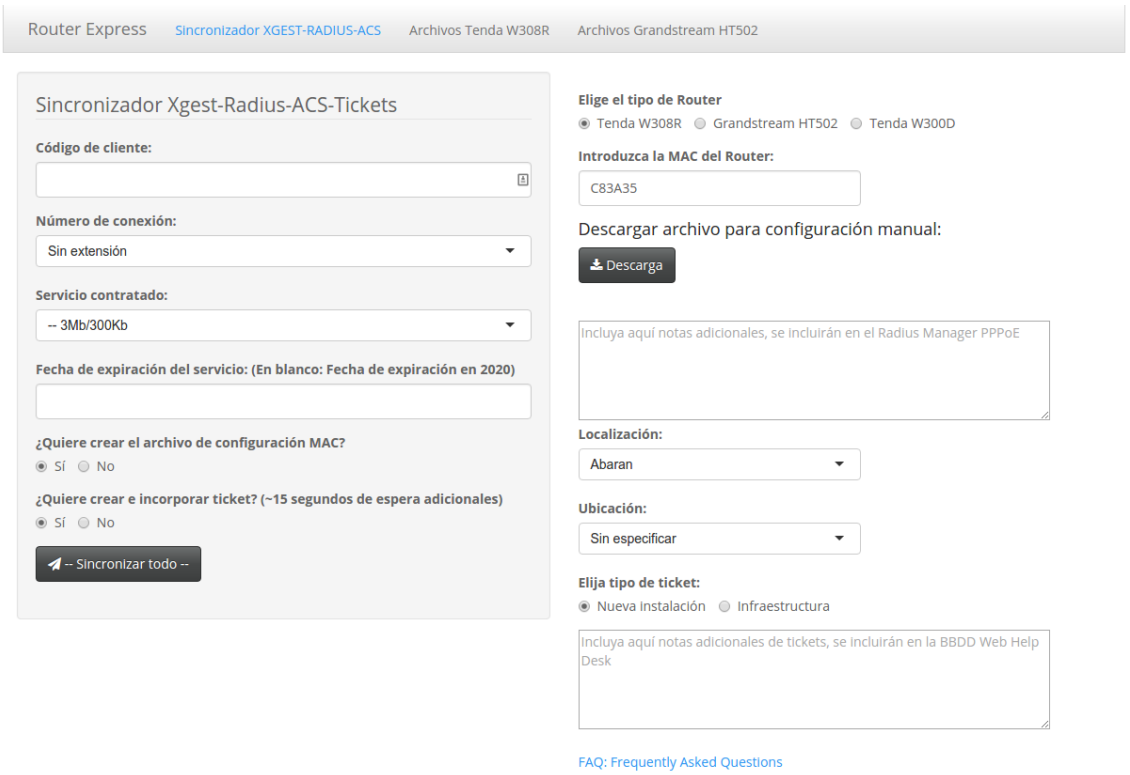

Figura 51: Router Express con todas las opciones de configuración de router y tickets activadas

Imaginemos que Juan González, un cliente imaginario que ya tiene conexión con Electrónica Martínez y número de teléfono, quiere una segunda conexión en su casa de playa, ubicada en una zona no residencial en la Manga. Quiere una conexión de 10 Megabytes con servicio de telefonía incluido en esa casa también, para 3 meses.

Como el usuario ya tiene conexión con Electrónica Martínez, sólo tendremos que buscar su número de cliente en la base de datos de clientes. Imaginemos que es el cliente 14907. Así que primero introduciremos el primer campo, el de código de cliente. En este pondremos el número 14907. En el campo de Número de conexión, elegiremos un "2", ya que como hemos dicho, este cliente dispone ya de una primera conexión con nosotros.

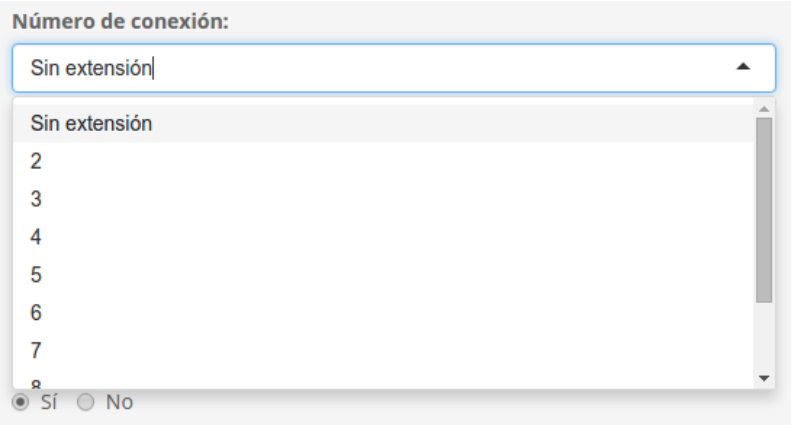

Figura 52: Eligiendo el número de conexión

Tras esto, nos aparecerán más opciones que rellenar:

Figura 53: Opciones que aparecen al escoger una conexión diferente a la primera

Donde rellenaremos la dirección, la ciudad y la región. Tras esto, escogeremos el servicio contratado. Como el cliente nos ha pedido 10Mb, le otorgaremos el servicio de 10Mb/1Mb.

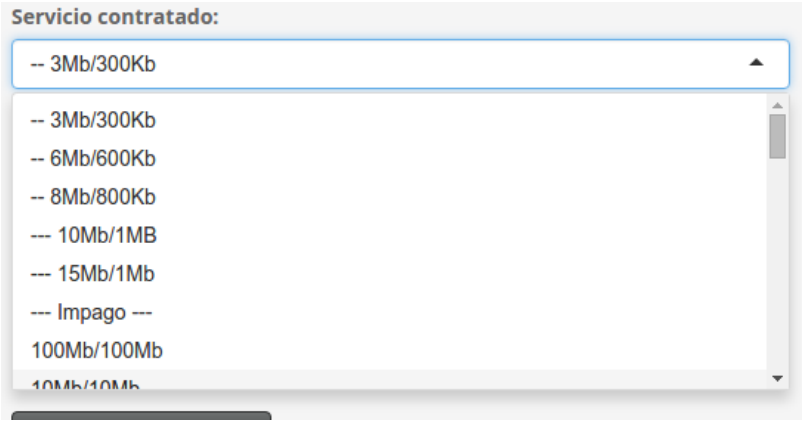

Figura 54: Elección de servicio contratado

Ahora tendremos que elegir la fecha de expiración, así que tendremos que darle click al recuadro de fecha y elegir una fecha. En este caso, lo pondríamos para tres meses contando a partir de hoy.

Podemos ver el widget en acción en la siguiente imagen:

Figura 55: Selección de fecha de expiración

A continuación, nos pregunta si queremos crear un archivo MAC, a lo que le diremos que sí. Instantáneamente nos aparecerá en el lado derecho estos widgets:

Figura 56: Creación de archivo MAC

Donde pondremos la MAC del router Tenda W308R que le pondremos al cliente. También dispone de una descarga manual del archivo por si queremos introducir el fichero a mano, por indisponibilidad de algún servicio.

Antes de continuar con la siguiente pregunta, queríamos notar que existe un cuadro de texto rellenable en el objeto derecho de la pantalla, donde se explica que esas notas se incluirán en el servidor RADIUS, que es el que le da Internet al cliente.

Lo siguiente que nos pregunta es si queremos introducir un nuevo ticket de instalación. Le daremos a “Sí”, ya que es una nueva instalación la que tendremos que realizar. Automáticamente nos saldrán las siguientes opciones que podemos ver:

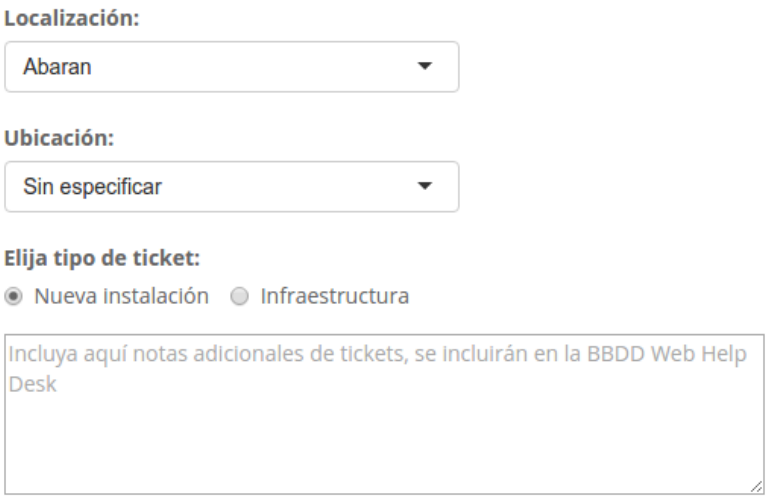

Figura 57: Opciones de entrada para el nuevo ticket.

Aquí tenemos varias opciones para elegir. Primero tendremos que elegir la localización para después elegir entre una lista de ubicaciones específicas. En nuestro caso escogeremos La Manga, y, como vemos en la siguiente foto, nos permite escoger entre varias ubicaciones en La Manga.

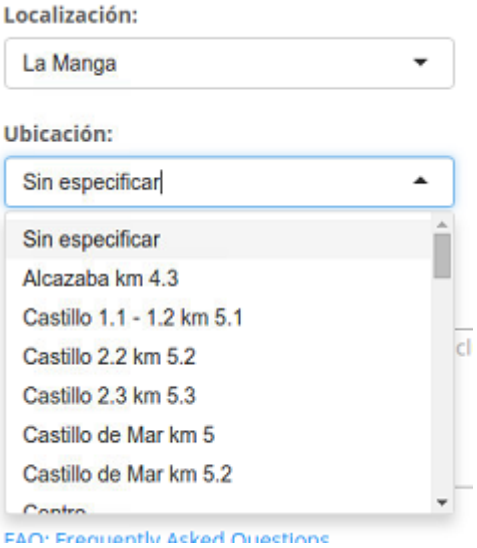

Figura 58: Ubicaciones en La Manga

Tras esto, podremos escoger entre catalogar este servicio como una Nueva instalación o una infraestructura, en el caso de despliegues en comunidades. También podemos insertar notas adicionales para el nuevo ticket que se va a crear, más abajo.

A continuación del recuadro, nos encontramos con el FAQ de la aplicación. Si le clickamos, nos mostrará un archivo en HTML creado con RMarkdown, para explicar el uso de Router Express.

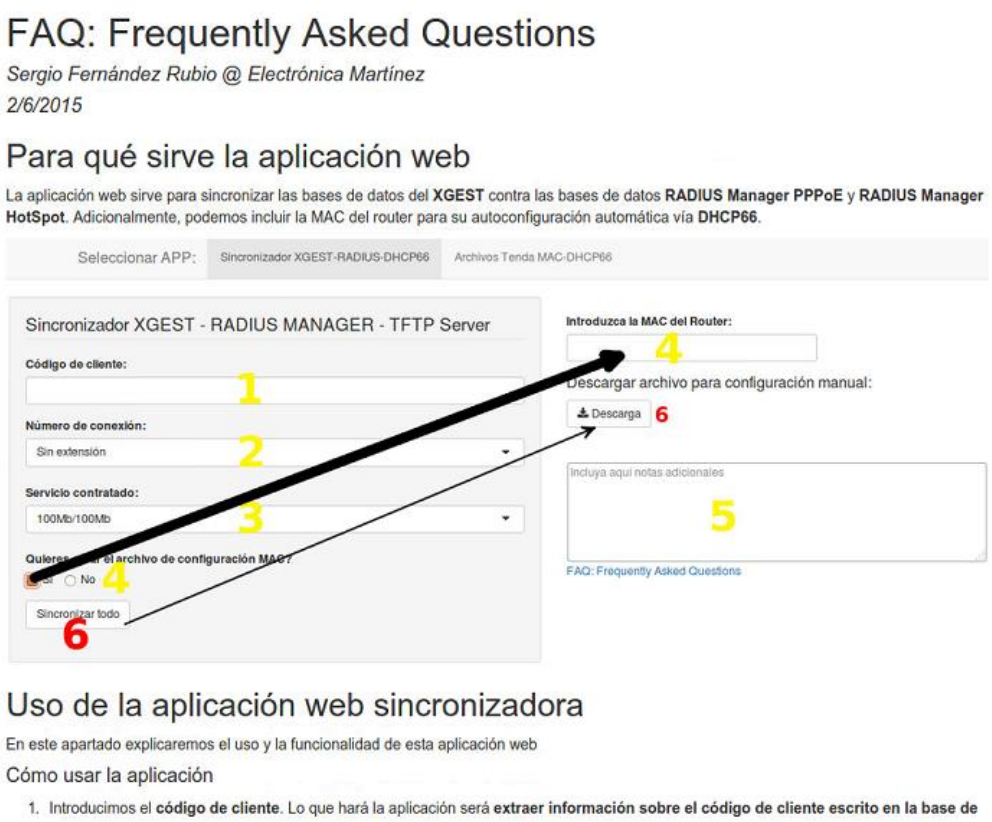

Figura 59: Vista previa del FAQ

Una vez tengamos todo rellenado, haremos click en “Sincronizar Todo”.

Cuando le presionemos, es de resaltar dos cosas:

- El botón se desactiva temporalmente hasta la finalización del proceso, esto es debido a que, si lo presionamos dos veces seguidas, el programa se ejecutará dos veces.
- Una barra superior se irá cargando, e irá indicando por qué fase de la aplicación va ejecutando, como vemos en la siguiente imagen:

Figura 60: Barra de carga.
Apreciamos la barra superior de indicación, y a la derecha una descripción de la fase del script.

Una vez finalizada la carga, tendremos varios indicadores:

Figura 61: Resultados visibles de la aplicación

La aplicación nos mostrará si el resultado ha sido correcto o no, según lo indicado en la parte inferior izquierda de la imagen. A la derecha obtendremos un texto formateado para ser enviado a los técnicos en caso de que fuera necesario.

Pero aún nos falta crear el archivo del router de telefonía. Para ello, procederemos a recargar la página para volver a la página inicial.

Tendremos que poner, esta vez, el número de cliente y su extensión, su localización la fecha de expiración, y como servicio elegiremos "Telefonía". Le daremos a que queremos crear el archivo de configuración MAC, y seleccionaremos el Grandstream HT502. Cuando lo elijamos, nos saldrán los siguientes widgets:

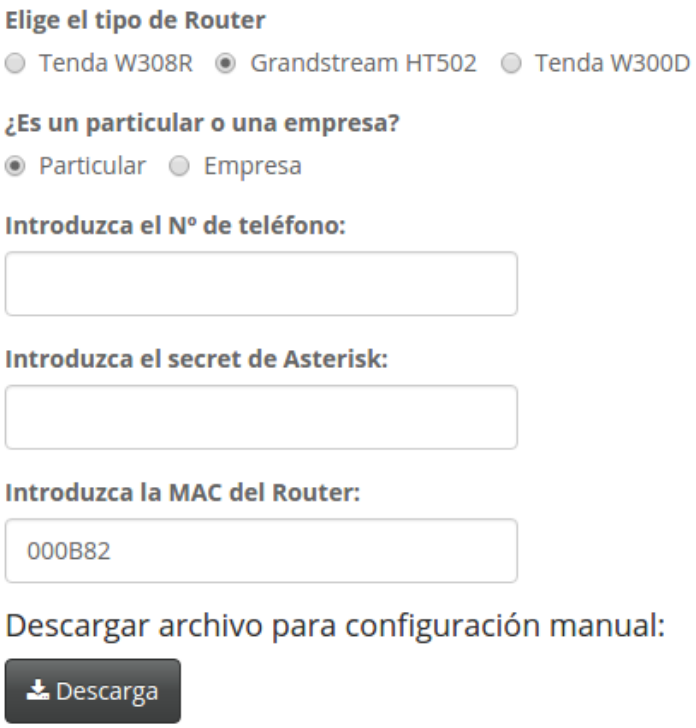

Figura 62: Widgets adicionales al escoger el router Grandstream HT502

Aquí tendremos que elegir si es un particular o una empresa, en este caso escogeremos un particular. Escogeremos el número de teléfono del cliente, luego su contraseña que obtendremos de la centralita, y pondremos su MAC.

Tras esto, le daremos a Sincronizar todo, y ya tendremos nuestro cliente PPPoE de telefonía creado, y nuestro archivo de configuración para el Grandstream también.

En los demás tabs, nos encontraremos con servicios de configuración para los archivos de DHCP66. En primer lugar, nos encontraremos con los archivos de configuración para los Tenda W308R.

Figura 63: Tab para la creación manual de archivos de configuración Tenda W308R

Aquí tenemos que introducir el código de cliente, las iniciales del cliente (para que la SSID del WiFi se modifique). También tendremos que introducir el usuario PPPoE y su contraseña, y también la MAC del router.

Cuando finalicemos, presionamos “Subir al servidor TFTP”, tras lo cual mostrará un mensaje de “Correcto”.

La siguiente tab, se utiliza para crear los archivos de configuración XML de los routers de voz Grandstream HT502.

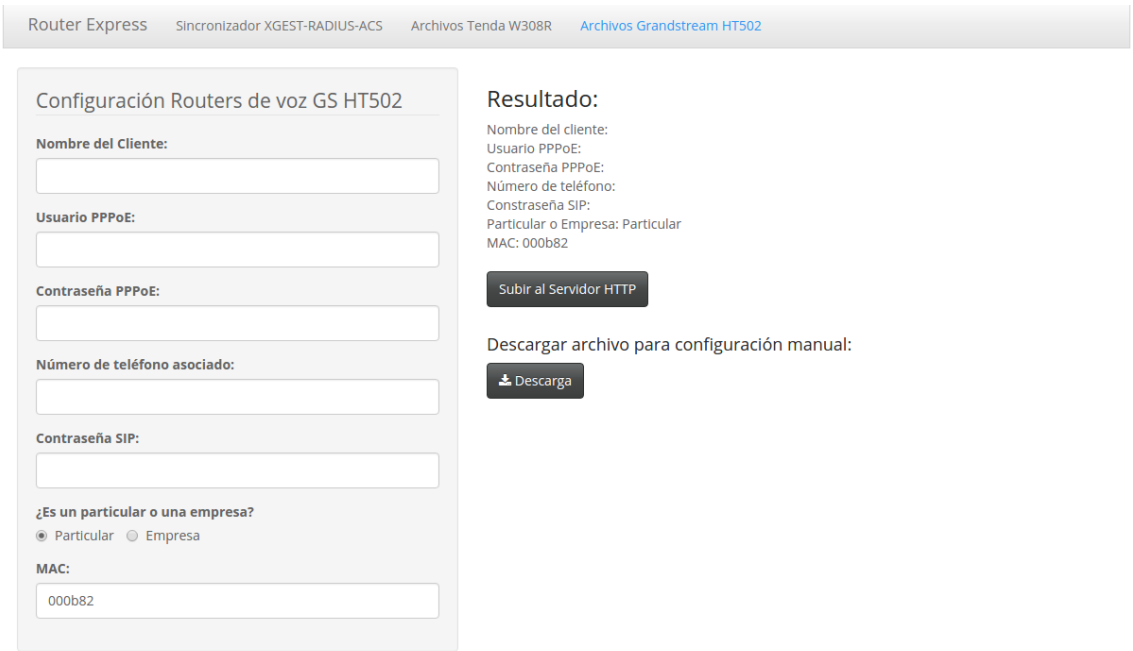

Figura 64: Tab para la creación manual de archivos de configuración Grandstream HT502

4.2. Presentación y explicación de código R

Comenzaremos explicando lo que podemos ver en la aplicación web que hemos creado.

Pero antes, necesitaremos iniciar el servidor web, y para ello usaremos la siguiente línea de código:

```
shiny::runApp(appDir = "/home/user/WebApp", launch.browser =
        FALSE, port = 9080, host = "0.0.0.0")
```

La expresión *shiny::* es una referencia al paquete shiny. Así no necesitaremos previamente cargar el paquete shiny en la aplicación. Usamos la función *runApp()* para lanzar la aplicación por un puerto, especificándole argumentos como los que vemos:

- El directorio donde se encuentra la aplicación especificado como *appDir*
- En segundo lugar, *launch.browser* sirve para decirle a shiny que no debe automáticamente lanzar un navegador web para visualizar la página. Lo desactivamos porque tras arrancar una y otra vez, no interesa que salga el navegador.
- Lo siguiente es el puerto donde shiny escuchará peticiones. Por regla general, sólo se pueden puertos por encima del 1024, ya que esos puertos están reservados.
- Por último, haciendo a modo de firewall, le indicamos la opción de quién puede acceder a la aplicación a través de la IP. Si le especificamos *0.0.0.0* quiere decir que escuchará todas las peticiones de cualquier IP.

Archivo ui.R

Como comienzo de la UI de nuestra página web, tenemos el siguiente código que procederemos a explicar:

```
shinyUI(fluidPage(
  navbarPage(title="Router Express", id="navbar", position="static-
        top", inverse=F, theme = "bootstrap.css",
        tabPanel("Sincronizador XGEST-RADIUS-ACS", value=1,
                fluidRow(sidebarPanel(width = 6, ...
```

Tras iniciar la UI con el comando *shinyUI()*, le expresamos mediante un *fluidPage()* y su interno *navbarPage()* que deseamos una página dividida por tabs. Necesitamos esto porque de este modo, daremos cabida a las dos aplicaciones complementarias de creación de archivos manuales para los router Tenda W308R y Grandstream HT502.

Como título de esa barra de navegación, tenemos el nombre de nuestra aplicación, llamada *Router Express*. Le asignaremos una *id*, lo pondremos de forma que la barra de navegación se quede fija arriba con *position="static-top"*, y usaremos un tema bootstrap, escogido de una página de elementos bootstrap gratuitos [40].

Tras esto, nos encontramos con un *tabPanel()*, que define un contenedor de tab o panel, en el que podemos escoger un nombre y un valor (este valor es un simple identificador de tab actual). Tras esto, nos encontramos con un *fluidRow()* que será el que contenga los dos principales elementos de la página. El primer elemento es el *sidebarPanel()* en el que podremos escoger su anchura, que hemos definifo como 6. Tras esto, nos encontramos con los primeros widgets de la página.

```
tags$legend("Sincronizador Xgest-Radius-ACS-Tickets"),

textInput("client1", "Código de cliente: "),
selectInput("selectExt", "Número de conexión: ", c("Sin extensión" = 1, "2" =
        2, "3" = 3, "4" = 4, "5" = 5, "6" = 6, "7" = 7, "8" = 8, "9" = 9, "10"
        = 10)),

uiOutput("direccion1"),
uiOutput("direccion2"),
uiOutput("direccion3"),
uiOutput("service"),

dateInput('dateExp', label = 'Fecha de expiración del servicio: (En blanco:
        Fecha de expiración en 2020)', value = "", language = "es"),

radioButtons(inputId="select",label="¿Quiere crear el archivo de
        configuración MAC?", choices=list("Sí","No" ), selected="No", =TRUE),

radioButtons(inputId="selectTICKET", label="¿Quiere crear e incorporar
        ticket? (~15 segundos de espera adicionales)",
        choices=list("Sí","No"),selected="No",inline=T),

actionButton("sincronizar", "\t Sincronizar todo \t", icon("paper-plane")),
        br(), br(),
textOutput("errores"),textOutput("erroresHS")
), br(),
```

Lo primero que nos encontramos es una llamada especial a elementos de HTML, como es *legend*. Este será el encabezado de nuestro panel. Hay muchas formas de llamar a elementos HTML y una es mediante el comando *tag$.*

Lo siguiente que nos encontramos es un *textInput()*, o una entrada de texto, en el que precisa de dos valores: El identificador (que usaremos para retornar su valor) y el encabezado del recuadro de texto.

A continuación, nos encontramos con un selectInput, con su identificador y encabezado. Pero necesitamos un tercer campo, que será el de *choices*. En él escogeremos las distintas posibilidades que hay para escoger en el widget.

Tras esto, nos encontramos con cuatro *uiOutput()*. Estos *uiOutput()* son, por decirlo así, contenedores de widgets. Podremos poner cualquier widget que queramos, y tendremos que definir qué es y cuándo aparece en el lado del servidor. En este caso, definen los widgets que aparecerán si seleccionamos un número de conexión mayor o igual a dos. Servirá para indicar la dirección del nuevo cliente.

Lo siguiente es un *dateInput(),* que será para definir la fecha de expiración del servicio. Como argumentos nos pide su *id* y su encabezamiento, como siempre. Ahora nos pide el valor predefinido de fecha, en la que debemos de escoger la fecha que queremos que aparezca. Como queremos que nos aparezca el día de hoy, lo definimos como un string vacío. También en el siguiente argumento *language* podemos definir el lenguaje que usará R para la representación de los meses.

A continuación tenemos dos *radioButtons(),* que nos servirán para crear el archivo de configuración MAC y crear el ticket, respectivamente. Toman como argumentos nuevos *choices()*, que nos definirá los botones que podremos seleccionar. En nuestro caso tenemos las opciones Sí y No. En el siguiente argumento, *selected*, podemos elegir cuál opción será la elegida por defecto, en nuestro caso, No. Por último, podremos definir si queremos que las opciones estén en línea, o una encima de otra.

Nos encontramos ahora con un *actionButton()*, que será el que se encargará de arrancar el script. Como detalle, vemos que tiene un argumento *icon()*, en el que hemos escogido un icono de la librería de iconos de Font Awesome [41].

Por último nos encontramos con dos *textOutput()* que son unos contenedores de un formato de texto predefinido. Aquí, desde el servidor, mandaremos los mensajes de resultado de las operaciones.

```
mainPanel(width = 5,
                    uiOutput("macSelect"),
                    uiOutput("telefono1"),
                    uiOutput("particularEmpresas"),
                    uiOutput("telefono2"),
                    uiOutput("pass_SIP"),
                    uiOutput("mac1"),
                    uiOutput("mac2"),
                    uiOutput("mac3"), br(),
                    uiOutput("macOK"), br(),
                    tags$textarea(id="CampoNotas", rows=5, cols=60,
                                  placeholder="Incluya aquí notas adicionales,
        se incluirán en el Radius PPPoE", value = ""), br(),
                    uiOutput("ticketLocalizacion"),
                    uiOutput("ticketUbicacion"),
                    uiOutput("TICKETtipo"),
                    uiOutput("CampoNotasTicket"),  br(),
                    a("FAQ: Frequently Asked Questions", target="_blank",
        href="http://X.X.X.X:8787/files/WebApp/FAQ/FAQ.html"), br(), br(),
                    uiOutput("numeroTICKET"),
                    uiOutput("TextoOut")
          )
        )

    ),
```

Aquí empieza el *mainPanel()*, que es el que se situará a la derecha de la página web. Dentro de ella, podremos establecer su anchura.
Lo que nos encontramos por todo el script son unos contenedores *uiOutput()*, que variarán dependiendo de qué elijamos en los selectores de MAC.

Es de notar que usamos uno de los *tags* que invoca al tag de HTML *textarea*, en el que insertaremos comentarios en la base de datos del servidor RADIUS.

También comentar la función *a()* que inserta un hipervínculo hacia el HTML generado por RMarkdown. Coge como argumentos el string de texto que aparecerá. También toma *target="_blank"* que quiere decir que una vez que pinchemos el enlace, nos abrirá otra pestaña en el navegador.

También pondremos el código de las dos pestañas restantes haciendo mención de los elementos que llamen la atención.

```
tabPanel("Archivos Tenda W308R", value=2,

      sidebarLayout(
        sidebarPanel(width = 5,

           tags$legend("Configuración Routers Tenda W308R"),
           textInput("client", "Código de cliente (Para la extensión de
        contraseña WiFi): "),
           textInput("initials", "Iniciales del cliente (Para el SSID WiFi):
        "),
           textInput("user", "Usuario PPPoE: "),
           textInput("pass", "Contraseña PPPoE: "),
           textInput("mac", "MAC del Router a instalar (Importante que sea
        correcto): ", value="C83A35")
                    ),

         mainPanel(width = 5,
           h3("Resultado: "),
           textOutput("client"),
           textOutput("initials"),
           textOutput("user"),
           textOutput("pass"),
           textOutput("mac"), br(),
           actionButton("upload", "Subir al Servidor TFTP"), br(), br(),
           textOutput("ok"),
           h4("Descargar archivo para configuración manual: "),
           downloadButton("downloadData", "Descarga")
           )
       )

    ),
```

Como cosa destacable, podemos observar el *textInput()* que tiene por *id,* "mac", presenta un argumento llamado *value* con valor "C83A35". Esto nos sirve para poner por defecto un valor en ese campo. Nos sirve para los ficheros de MAC, ya que cada marca tiene una misma OUI (las 6 primeras cifras hexadecimales de una MAC).

Otra cosa para destacar es el widget *downloadButton().* Este botón hará que cuando lo pinches, puedas descargar el archivo de configuración manualmente.

Siguiente tab:

```
tabPanel("Archivos Grandstream HT502", value=3,

             fluidRow(
               sidebarPanel(width = 5,

                 tags$legend("Configuración Routers de voz GS HT502"),
                 textInput("nombreGS", "Nombre del Cliente: "),
                 textInput("userGS", "Usuario PPPoE: "),
                 textInput("passGS", "Contraseña PPPoE: "),
                 textInput("telefonoGS", "Número de teléfono asociado: "),
                 textInput("passSIPGS", "Contraseña SIP: "),
                 radioButtons(inputId="selectParticularEmpresaGS", label="¿Es
                       un particular o una empresa?",
                       choices=list(`Particular","Empresa"),
                       selected="Particular", inline=TRUE),
                 textInput("macGS", "MAC: ", value="000b82")

               ),

               mainPanel(width = 5,
                         h3("Resultado: "),
                         textOutput("nombreGS"),
                         textOutput("userGS"),
                         textOutput("passGS"),
                         textOutput("telefonoGS"),
                         textOutput("passSIPGS"),
                         textOutput("selectParticularEmpresaGS"),
                         textOutput("macGS"), br(),
                         actionButton("uploadGS", "Subir al Servidor HTTP"),
       br(), br(),
                         textOutput("okGS"),
                         h4("Descargar archivo para configuración manual: "),
                         downloadButton("downloadDataGS", "Descarga")
               )
             )
    )
  )

))
```

Lo único a señalar es que en este caso, el *textInput()* que tiene como *id* "macGS", tiene como valor predeterminado "000b82". Esto es porque ha cambiado el OUI de la empresa, ya que ahora no es Tenda sino Grandstream. También es de señalar que para los archivos xml, es importante que sus letras estén en minúsculas. De otro modo, el router no podría coger el archivo de su MAC.

Ya hemos terminado de examinar el código referente a la parte de la UI de Shiny. Procedamos ahora con el código de servidor, más extenso y complejo.

Archivo server.R

Este archivo es el que ejecutará el servidor cada vez que un cliente inicie sesión en la página web.

El código empieza cargando a memoria los paquetes necesarios, que luego utilizaremos en nuestro script.

```
library(shiny)
library(RMySQL)
library(digest)
library(stringr)
library(XML)
```

El primero ya lo hemos comentado. El segundo paquete, *RMySQL* es el paquete de conectividad con bases de datos MySQL, que usaremos para las bases de datos RADIUS y la base de datos de clientes.

También usaremos la librería *digest*, que es la que nos ayudará a generar las contraseñas de los clientes PPPoE en un formato que preferimos no mencionar.

Y por último usaremos el paquete *stringr* que es el paquete por excelencia para trabajar con strings de texto. También cargaremos la librería XML para trabajar con archivos en formato XML.

```
options(shiny.trace=TRUE)

limpiaMAC <- function(x) return( str_replace_all( str_replace_all(
        str_replace_all(x, ":", ""), "-", ""), " ", "") )

limpiaESPACIOS <- function(x) return(str_replace_all(x, " ", ""))
```

Lo siguiente que nos encontramos es con una opción para activar los logs en la consola, o el medio que estemos utilizando como lanzador de la webapp en shiny.

Tras esto, hemos creado dos funciones que nos serán muy útiles para trabajar más fácilmente. Esta primera función nos servirá para limpiar contenidos de la MAC, ya que al introducir la MAC, podemos insertar caracteres como “:”, “-”, ó “ ”. Esta primera función nos servirá para eso, mientras que la segunda nos limpiará solamente los espacios, utilizada para limpiar datos que cogeremos de la base de datos de clientes.

En el siguiente bloque de código (que tenemos en la siguiente página) empieza la declaración de nuestro servidor Shiny, en función de *input* y *output*, que es la manera en que tiene Shiny de comunicar servidor y cliente.

Tras esto, nos encontramos con una expresión reactiva. Nos encontraremos con muchas más en los siguientes bloques de texto. Sirven como *output* para la UI del servicio de la aplicación web. Esto es debido a que el widget que queremos introducir

requiere de una computación previa, que procederemos a explicar ahora.

```
shinyServer(function(input, output) {
  output$service <- renderUI({
      radius <- dbConnect(MySQL(), user="user", password="pass", db="radius",
        host="X.X.X.X")
      rm_services <- dbGetQuery(radius, "select srvid, srvname from
        rm_services")
      dbDisconnect(radius)

      rm_services <- rm_services[!row.names(rm_services) %in% grep("^srvid",
        rm_services$srvname),]
      choices <- setNames(rm_services$srvid, rm_services$srvname)

      selectInput("service", "Servicio contratado: ", choices)
  })
```

Dento de *renderUI()*, observamos la función *dbConnect()*, que será la que cree la conexión con la base de datos RADIUS, con el fin de extraer los diferentes perfiles de velocidad que podrá escoger el cliente.

Lanzamos tras esto, un query, escogiendo dos columnas: Una de ellas será el identificador de servicio, y la otra será el nombre que tiene. Tras esto, nos desconectaremos de la base de datos.

Luego, quitaremos del *data frame* extraído, los servicios cuyo nombre empiece por "srvid", ya que éstos, no tienen asociado ningún perfil de velocidad. Para ello, usaremos la función *grep()*, que será la encargada de encontrarlos, y le aplicaremos el comando %in% para que donde existan, coja su índice y sólo coja los que no están en ese índice. Es una línea algo compleja, pero básicamente limpia los servicios que se llaman "srvid".

Tras esto, debemos hacer un formato de esas dos columnas para que cuando seleccionemos un servicio contratado, se vea el nombre visualmente, y dentro de la aplicación, veamos el identificador. Para ello usamos la función *setNames()* sobre las dos columnas.

Por último, declaramos nuestro widget: Un *selectInput()* con el que podremos seleccionar el servicio que haya contratado el cliente.

```
output$direccion1 <- renderUI({
    if(as.numeric(input$selectExt) >= 2){
      textInput("direccion11", "Dirección de la instalación: ")
    }})
output$direccion2 <- renderUI({
    if(as.numeric(input$selectExt) >= 2){
      textInput("direccion22", "Ciudad: ")
    }})
output$direccion3 <- renderUI({
    if(as.numeric(input$selectExt) >= 2){
      textInput("direccion33", "Región: ")
    }})
```

En el anterior bloque de texto, tenemos los tres *renderUI()* por si escogemos una extensión para un número de cliente mayor a dos, introducir la nueva dirección. Son reactivos porque deben aparecer sólo si seleccionan una extensión mayor o igual a dos.

```
output$macSelect <- renderUI({
  if (input$select == "No") return()

  else radioButtons(
     inputId="selectRouter", label="Elige el tipo de Router",
      choices=list("Tenda W308R", "Grandstream HT502", "Tenda W300D"),
      selected="Tenda W308R", inline=TRUE)
  })

output$mac1 <- renderUI({
  if (input$select == "No")
    return()

  else if (input$selectRouter == "Tenda W308R"  &&
      !is.null(input$selectRouter)) {
    textInput("MAC_1", "Introduzca la MAC del Router: ", value="C83A35")}
  else if (input$selectRouter == "Grandstream HT502" &&
      !is.null(input$selectRouter)) {
    textInput("MAC_1", "Introduzca la MAC del Router: ", value="000B82"}
  else if (input$selectRouter == "Tenda W300D" &&
      !is.null(input$selectRouter)) {
    textInput("MAC_1", "Introduzca la MAC del Router: ", value="C83A35")}
})

output$mac2 <- renderUI({
  if (input$select == "No" ) return()

  else if(!is.null(input$selectRouter))
    if (input$selectRouter == "Tenda W300D")
      return()

  else h4("Descargar archivo para configuración manual: ")
})

output$mac3 <- renderUI({
  if (input$select == "No" ) return()

  else if(!is.null(input$selectRouter))
    if (input$selectRouter == "Tenda W300D") return()

  else downloadButton("downloadData2", "Descarga")
})

output$downloadData2 <- downloadHandler(
  filename = function() {
    if(input$selectRouter == "Grandstream HT502") MAC_GS else MACapp
  },
  content = function(file) {
    write(archivo, file)
  }
)

output$macOK <- renderUI({
  textOutput("erroresMAC")
})
```

En este gran bloque de texto, tenemos asociado la funcionalidad de la creación de los archivos MAC, si le pulsamos que sí a la opción de crear archivos.

Tenemos primero unos *radioButton()* en los que escogeremos el router que deseamos autoconfigurar, y tras esto, en el siguiente *renderUI()*, tres posibles *textInput()*, ya que dependiendo del router que escojamos se nos mostrará un textInput con una ID diferente.

En los tres siguientes *renderUI()* tendremos la opción para descargar el archivo de configuración, no apareciendo esa opción si escogemos el router Tenda W300D, ya que éste funciona por TR069.

Es de destacar el widget *downloadHandler()*, que necesita dos argumentos. Primeramente, el nombre del archivo a descargar, y segundo, sus datos (o fichero).

En el último *renderUI* nos encontramos con el mensaje de error de los archivos de configuración, en la que nos avisará si ha habido algún error.

```
output$particularEmpresas <- renderUI({
    if (input$select == "Sí" && !is.null(input$selectRouter)){
      if (input$selectRouter == "Grandstream HT502"){

        radioButtons(inputId="selectParticularEmpresa", label="¿Es un
       particular o una empresa?", choices=list( "Particular","Empresa"),
        selected="Particular", inline=TRUE)

      }
      else return()
    }
    else return()
  })

  output$telefono2 <- renderUI({
    if(as.numeric(input$service) == 23  && !is.null(input$service)){
      textInput("telefono", "Introduzca el N° de teléfono: ")
    }
    else if (input$select == "Sí"  && !is.null(input$selectRouter)){
      if (input$selectRouter == "Grandstream HT502"){
          textInput("telefono", "Introduzca el N° de teléfono: ")
      }
      else return()
    }
    else return()
  })

  output$pass_SIP <- renderUI({
    if (input$select == "Sí"  && !is.null(input$selectRouter)){

      if (input$selectRouter == "Grandstream HT502")
          textInput("passSIP", "Introduzca el secret de Asterisk: ")

        else return()

    }
  })
```

En el anterior bloque de código, tenemos la funcionalidad para cuando escogemos un router Grandstream HT502. Cuando clickamos en él, nos aparecerán 3 widgets.

El primero, nos dará a elegir entre si lo vamos a usar para un particular, o si lo vamos a usar en una empresa. Esto es simplemente un parámetro de la IP a la que deberá de conectarse el Grandstream HT502, ya que en la empresa se hace una diferenciación entre la centralita de empresa, y la centralita de particulares.

El siguiente widget será para la introducción del número telefónico asociado a ese Grandstream. Es de destacar que este widget se activará por defecto si seleccionamos crear un usuario PPPoE con servicio de telefonía, ya que es necesario introducirlo en la base de datos RADIUS.

Por último, en el último *renderUI(),* tendremos que poner la contrseña que nos dará la centralita.

En la siguiente página, nos encontraremos con el código de los widgets que aparecen al querer sincronizar con la base de datos de tickets.

Lo primero que nos encontramos es un *renderUI()*, que en su interior alberga un tag HTML de *textarea*. En él, podremos incluir notas adicionales con las que luego trabajarán los empleados. Es de notar que, como explicaremos más adelante, se adhiere a este campo de notas los usuarios y contraseñas PPPoE, así como su usuario HotSpot.

El siguiente widget nos dejará escoger entre Infraestructura o Nueva instalación. Tras esto, tendremos dos *renderUI()* que llamarán a la base de datos de tickets en PostgreSQL, para hallar las localizaciones y ubicaciones posibles, insertadas dentro de la base de datos.

Para conseguirlas, primero cargaremos el paquete *RPostgreSQL*, crearemos un *driver* para esa base de datos, y usaremos la conocida función *dbConnect()*. Tras recibir los datos, debemos procesarlos porque hay localizaciones que no existen, que empiezan por 0. Para ello llamaremos a la función *grepl()*, con la que nos dará un vector de posiciones donde se encuentren esos valores. Por ello, sólo nos quedaremos con los valores que no coincidan con los resultados de *grepl()*.

Básicamente, ambos *renderUI()* tienen un mismo patrón, pero notamos que el segundo depende del *input* del otro. Tras recoger los resultados de las ubicaciones, sólo nos quedaremos con las ubicaciones que coincidan con nuestra localización elegida en el *renderUI()* anterior. Añadiremos luego, otra opción más “Sin especificar”, para cuando no deseemos especificar ninguna ubicación dentro de esa localización o no exista ninguna ubicación para la localización elegida.

```
output$CampoNotasTicket <- renderUI({
    if(input$selectTICKET == "No"){
      return()
    }
    else{
      tags$textarea(id="CampoNotasTICKETS", rows=5, cols=60,
                    placeholder="Incluya aquí notas adicionales de tickets, se
        incluirán en la BBDD",
                    value = "")
    }
  })

  output$TICKETtipo  <- renderUI({
    if(input$selectTICKET == "No"){
      return()
    }
    else{
      radioButtons(
        inputId="tipoTICKETui", label="Elija tipo de ticket: ", choices=list(
          "Nueva instalación", "Infraestructura"), selected="Nueva instalación",
        inline=T)
    }
  })

  output$ticketLocalizacion <- renderUI({
    if(input$selectTICKET == "No"){
      return()
    }
    else{
      library("RPostgreSQL")
      drv <- dbDriver("PostgreSQL")
      con <- dbConnect(drv, user="user", password="pass", dbname="tickets",
        host="XX.XX.XX.XX", port=9999)
      query <- dbGetQuery(con, "SELECT location_name, location_id FROM location
        ORDER BY location_name")
      dbDisconnect(con)
      query <- query[!grepl("^0", query[[1]]),]
      choices <- setNames(query[,2], query[,1])
      selectInput("ticketLocalizacion1", "Localización: ", choices)
    }
  })

  output$ticketUbicacion <- renderUI({
    if(input$selectTICKET == "No"){
      return()
    }
    else{
      library("RPostgreSQL")
      drv <- dbDriver("PostgreSQL")
      con <- dbConnect(drv, user="user", password="pass", dbname="tickets",
        host="XX.XX.XX.XX", port=9999)
       queryUBI <- dbGetQuery(con, "SELECT location_id, room_name FROM room ORDER BY
        room_name")
       dbDisconnect(con)
       queryUBI <- queryUBI[queryUBI[,1] == input$ticketLocalizacion1, 2]
       queryUBI <- c("Sin especificar", queryUBI)
       selectInput("ticketUbicacion1", "Ubicación: ", queryUBI)
    }
  })
```

Ahora empezaremos a explicar la siguiente etapa del archivo de servidor: Tras pulsar el botón de Sincronizar.

```
observeEvent(
  input$sincronizar,{

    if(input$client1 != ""){

      disable("sincronizar")

      withProgress(message = 'Ejecutando R-Express', value = 0, {
```

Este *observeEvent()* escuchará los cambios en el elemento siguiente que le pongamos (en nuestro caso, el botón de Sincronizar), y ejecutará el código que le pongamos.

La ejecución comienza comprobando si el *textInput()* del cliente está vacío, para así no seguir la ejecución. Posteriormente deshabilitamos el botón de Sincronizar, e inicializamos la barra de progreso superior que nos hará saber en qué parte de la ejecución se encuentra el programa.

```
incProgress(1/6, detail = "Llamando BBDD")

print(Sys.time())
print("Botón pulsado, llamando a BBDD...")

client <- dbConnect(MySQL(), user="user", password="pass",
        db="clientes", host="XX.XX.XX.XX", port=9999)
usuario_elegido <- dbGetQuery(client, sprintf("select * from
        fccli001 WHERE CCODCL = '%s'", input$client1))

Encoding(usuario_elegido$CDOM) <- "latin1"
Encoding(usuario_elegido$CNOM) <- "latin1"

dbDisconnect(xgest)
```

En este bloque de código extraeremos datos fundamentales de la base de datos principal de clientes. Lo primero que hacemos es incrementar la barra de progreso superior, tras lo cual imprimimos en consola la fecha y un mensaje de log.

En la consulta, elegimos sólo que nos devuelva los datos para ese usuario que hemos puesto, tras lo cual, cambiamos de codificación para que no haya problemas al importar los datos del cliente. Es de notar que no hace falta que precarguemos el paquete RMySQL, ya que lo hemos hecho al principio del código.

En las siguientes dos páginas nos encontraremos el código que hace la llamada a RADIUS PPPoE para insertar los datos en la base de datos.

Esta vez también incrementamos la barra de progreso, tras lo cual, nos conectaremos a la base de datos RADIUS. Efectuaremos dos querys, en los que sacaremos previamente, datos de dos tablas.

```
incProgress(2/6, detail = "Llamando a RADIUS ")

print("Llamando a Radius PPPoE")

radius <- dbConnect(MySQL(), user="user", password="PASS", db="radius",
        host="XX.XX.XX.XX")
radcheck <- dbGetQuery(radius, "select username, id from radcheck ORDER BY id")
rm_users <- dbGetQuery(radius, "select username from rm_users")

numeroCliente <- input$client1
if(as.numeric(input$selectExt) >= 2){
   numeroCliente <- paste(as.numeric(input$client1), as.numeric(input$selectExt),
        sep = "e")}
if(as.numeric(input$service) == 23){
   numeroCliente <- paste0(numeroCliente, "t")}

numeroClienteSinExtension <- input$client1

if (!(numeroCliente %in% radcheck$username)){
    passPlain <- paste(sample(c(0:9, letters[-12], LETTERS[-9]), size=8,
        replace=TRUE), collapse="")
    dbSendQuery(radius, sprintf("insert into radcheck (id, username, attribute, op,
        value) values (%i, '%s', 'Cleartext-Password', ':=', '%s')",
        tail(radcheck$id, n=1)+1, numeroCliente, passPlain))
    dbSendQuery(radius, sprintf("insert into radcheck (id, username, attribute, op,
        value) values (%i, '%s', 'Simultaneous-Use', ':=', '1')", tail(radcheck$id,
        n=1)+2, numeroCliente))
}
else{
     passPlainExistent <- dbGetQuery(radius, sprintf("select attribute, value from
        radcheck WHERE username = '%s'", numeroCliente))
     passPlainExistent <- passPlainExistent[passPlainExistent$attribute ==
"Cleartext-Password",]
     passPlain <- passPlainExistent$value
}

passCipher <- digest(passPlain, algo="XXX", ascii = X, serialize = X)
if(as.character(input$dateExp) == Sys.Date()) dateExp <- "2020-01-01"
else dateExp <- as.character(input$dateExp)

            username <- numeroCliente
            password <- passCipher
            enableuser <- 1
            firstname <- usuario_elegido$CNOM
            lastname <- if(as.numeric(input$service) == 23) input$telefono else ""
            phone <- usuario_elegido$CTEL1
            mobile <- usuario_elegido$CTEL2
            address <- usuario_elegido$CDOM
            city <- usuario_elegido$CPOB
            zip <- usuario_elegido$CCODPO
            country <- "Spain"
            state <- usuario_elegido$CPAIS
            comment <- sprintf("Contrasena: %s\n%s", passPlain, input$CampoNotas)
            expiration <- sprintf("%s 00:00:00", dateExp)
            srvid <- input$service
            createdon <- Sys.Date()
            createdby <- "sincronizador"
            owner <- "sincronizador"
```

```
if(as.numeric(input$selectExt) >= 2){
   address <- input$direccion11
   city <- input$direccion22
   state <- input$direccion33
}

if(!(numeroCliente %in% rm_users$username)){
   dbSendQuery(radius, sprintf("INSERT INTO rm_users (username, password,
        enableuser, firstname, lastname, phone, mobile, address, city, zip, country,
        state, comment, expiration, srvid, createdon, createdby, owner) SELECT '%s',
        '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s',
        '%s', '%s', '%s', '%s', '%s' FROM dual WHERE NOT EXISTS (SELECT 1 FROM
        rm_users WHERE username='%s')", username, password, enableuser, firstname,
        lastname, phone, mobile, address, city, zip, country, state, comment,
        expiration, srvid, createdon, createdby, owner, username))

    output$errores <- renderText("Correcto. El usuario no existía en Radius Manager
        PPPoE")

}
else{
    output$errores <- renderText("Error. El usuario ya existe en Radius Manager
        PPPoE")
}
dbDisconnect(radius)
```

La tabla radcheck la usaremos para comprobar si existe ya el usuario en la base de datos, con el fin de no sobreescribir. También nos servirá para saber qué ID tendremos que poner para no entrar con ninguna otra en uso. De otro modo tendríamos un conflicto entre IDs y la base de datos contendría errores.

La tabla rm_users simplemente la usaremos porque tendremos que insertar datos en esta tabla, para no sobreescribir tampoco.

Después de obtener datos de estas dos tablas, lo que haremos será formar el resto del número de cliente. Si es una segunda conexión o de valor más alto, le añadiremos una "e" entre el número de cliente y la extensión, y si es de telefonía, le añade una "t", simplemente para diferenciar.

A continuación, entraremos en el primer gran *if-else*. Este nos sirve para saber qué hacer, si existe o no existe el cliente en la tabla radcheck. Si no existe, generamos una contraseña para el PPPoE y la insertamos con los valores obtenidos de la base de datos de clientes mediante dos *dbSendQuery().* Si existe ya, simplemente cogeremos la password para su futuro uso.

Tras esto, salimos del *if-else* y calculamos manualmente la contraseña cifrada y la guardamos. Luego, hacemos una comprobación del valor de *input$dateExp*, que proviene del widget en el que escogemos la fecha de expiración. Por defecto, si no pones ningún valor, coge el día de hoy. Así que si el *input* nos devuelve el valor del día actual, eso quiere decir que la fecha de expiración será en 2020. Si hemos puesto algún dato en el widget, cogerá esa fecha.

Tras esto, hacemos varias asignaciones para ubicarnos mejor y utilizar la jerga de la base de datos RADIUS, nada que destacar.

Luego nos encontramos con un *if*, que comprueba que si es una conexión adicional ala primera, coge los datos de los widgets sobre direcciones que aparecen tras escoger un número igual a 2 o mayor en el campo de Número de conexión, y los inserta.

Pasamos a la siguiente página de código. En este *if-else* insertaremos todos los datos del usuario en la tabla rm_users. Para ello haremos uso de la sentencia "INSERT INTO x … SELECT … FROM dual WHERE NOT EXISTS (SELECT 1 FROM x WHERE username=x)".
Lo que hace esta sentencia es introducir datos, pero con la condición de no sobreescribir buscando en la misma base de datos el cliente escogido por si existe.

Si termina correctamente la instrucción, se generará un mensaje de aprobación, y si no, saldrá un mensaje de error. Finalmente, cerramos la conexión.

Lo siguiente que haremos será introducir datos en la base de datos RADIUS HotSpot, con el código que vemos en la siguientes dos páginas. Las bases de datos son muy parecidas, así que el código también lo será, sólo hay unos detalles que cambian, y que explicaremos a continuación.

Primeramente, hará una comprobación para hacer que, si el nuevo usuario es de telefonía únicamente, no cree un usuario HotSpot. Tras eso, se conectará a las bases de datos e igualmente leerá dos tablas.

El siguiente *if* nos comprobará que si lo que estamos creando es un cliente PPPoE con extensión, quiere decir que el usuario ya existe, con lo cual le haremos un UPDATE a su cuenta para poder usar X dispositivos más. Si es un nuevo cliente, entrará al *else* y creará el usuario al igual que hemos explicado antes.

El resto contiene unas leves modificaciones, y es al final donde queremos llegar. Allí notamos que hay una sentencia *observe()*. Esta es una expresión reactiva de Shiny, pero especial, ya que no dan lugar a un resultado, y por tanto no pueden ser usadas como un *input* de otras expresiones reactivas. La usaremos con el fin de hacer que funcionen las funciones reactivas *renderUI()*. Estas harán que en la interfaz web aparezca un widget en el que dentro se encontrará toda la información del cliente de las bases de datos PPPoE y Hotspot.

```
if(!(as.numeric(input$service) == 23)){

   incProgress(3/6, detail = "Llamando a RADIUS HOTSPOT")

   print("Llamando a Radius Hotspot")

   radiusHS <- dbConnect(MySQL(), user="user", password="pass", db="radius",
        host="XX.XX.XX.XX")
   rm_usersHS <- dbGetQuery(radiusHS, "select username from rm_users")
   radcheckHS <- dbGetQuery(radiusHS, "select username, id from radcheck ORDER BY
        id")

   if(as.numeric(input$selectExt) >= 2){
      numeroClienteSinExtension <- input$client1
      UsoSimultaneo <- as.numeric(input$selectExt)
      dbSendQuery(radiusHS, sprintf("UPDATE radcheck SET value = '%s' WHERE username
        = '%s' AND attribute = 'Simultaneous-Use'", UsoSimultaneo,
        numeroClienteSinExtension)) pass <- dbGetQuery(radiusHS, sprintf("SELECT
        value FROM radcheck WHERE username = '%s' AND attribute = 'Cleartext-
        Password'", numeroClienteSinExtension))
      pass <- pass[[1]]
    }
    else {
       passPosible <- X
       if (passPosible == "") passPosible <- X
       if (passPosible == "") pass <- X
       else pass <- limpiaESPACIOS(passPosible)

       if (!(numeroClienteSinExtension %in% radcheckHS$username)){
           dbSendQuery(radiusHS, sprintf("insert into radcheck (id, username,
               attribute, op, value) values (%i, '%s', 'Cleartext-Password', ':=',
               '%s')", tail(radcheckHS$id, n=1)+1, numeroCliente, pass))
           dbSendQuery(radiusHS, sprintf("insert into radcheck (id, username,
               attribute, op, value) values (%i, '%s', 'Simultaneous-Use', ':=',
               '1')", tail(radcheckHS$id, n=1)+2, numeroCliente))
                }
              }


              username <- numeroClienteSinExtension
              password <- digest(pass, algo="XXX", ascii = X, serialize = X)
              groupid <- 9
              phone <- usuario_elegido$CTEL1
              if(usuario_elegido$CTEL2=="") {
                mobile <- if(usuario_elegido$CTEL1 %in% c("0","6","7"))
                       usuario_elegido$CTEL1 else ""
              }

              comment <- sprintf("Contrasena: %s", pass)
              srvid <- 70
```

```
if(!(numeroClienteSinExtension %in% rm_usersHS$username)){
     dbSendQuery(radiusHS, sprintf("INSERT INTO rm_users (username, password,
        groupid, enableuser, firstname, phone, mobile, address, city, zip, country,
        state, comment, expiration, srvid, createdon, createdby, owner) SELECT '%s',
        '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s',
        '%s', '%s', '%s', '%s', '%s' FROM dual WHERE NOT EXISTS (SELECT 1 FROM
        rm_users WHERE username='%s')", username, password, groupid, enableuser,
        firstname, phone, mobile, address, city, zip, country, state, comment,
        expiration, srvid, createdon, createdby, owner, username))

      output$erroresHS <- renderText("Correcto. El usuario no existía en Radius
               Manager HotSpot")
}
else {
     if(as.numeric(input$selectExt) >= 2){
          output$erroresHS <- renderText(sprintf("Correcto. Pero el usuario ya
        existe en Radius Manager HotSpot así que no se ha añadido (Ahora el usuario
        puede usar %s dispositivos simultáneamente)", input$selectExt))
     }
     else{
          output$erroresHS <- renderText("Error. El usuario ya existe en Radius
        Manager HotSpot")
     }
}

dbDisconnect(radiusHS)
}

               observe({
                 output$TextoTelegram <- renderUI({
                   verbatimTextOutput("TextoTelegrama")
                 })


                 output$TextoTelegrama <- renderText({
                   if (as.numeric(input$service) != 23){
                     sprintf("PPPoE------------------ \nuser: %s \npass: %s
                         \nHotSpot---------------\nuser: %s \npass: %s",
                           numeroCliente, passPlain, numeroClienteSinExtension, pass)
                   } else {
                     sprintf("PPPoE------------------ \nuser: %s \npass: %s",
                             numeroCliente, passPlain)
                   }
                 })

             })
```

Lo siguiente con lo que nos encontramos es la parte a la llamada de la base de datos de tickets. El código estará en las dos páginas siguientes. Primero comprobaremos que cumplimos el primer *if*, es decir, que hayamos pulsado el botón de adjuntar ticket, tras lo cual incrementaremos el progreso y haremos un log.

Leeremos un archivo de tickets en csv descargado de la propia UI de la base de datos de tickets, que servirá para meter todos los datos necesarios en ese archivo y luego subirlo vía interfaz web.

```
if(input$selectTICKET == "Sí"){

   incProgress(4/6, detail = "BBDD Tickets")

   print("Llamando a BBDD Tickets")

   ticket <- read.csv("ticket.csv", check.names = F, na.strings = "", encoding =
        "ASCII")
   ticket[2] <- format(Sys.time(), "%d/%m/%y %R")
   ticket[7] <- input$client1

   library("RPostgreSQL")
   drv <- dbDriver("PostgreSQL")
   con <- dbConnect(drv, user="user", password="user", dbname="pass",
        host="XX.XX.XX.XX", port=9999)
   query <- dbGetQuery(con, "SELECT location_name, location_id FROM location ORDER
        BY location_name")
   dbDisconnect(con)
   ticket[8] <- query[query[,2] == input$ticketLocalizacion1, 1]

   if(input$tipoTICKETui == "Nueva instalación") ticket[9] <- "Instalaciones" else
        ticket[9] <- "Infraestructura"

   ticketNotaAdicional <- if (as.numeric(input$service) != 23){
        sprintf("PPPoE------------------ \nuser: %s \npass: %s
                 \nHotSpot---------------\nuser: %s \npass: %s", numeroCliente,
                passPlain, numeroClienteSinExtension, pass)
   } else {
         sprintf("PPPoE------------------ \nuser: %s \npass: %s", numeroCliente,
        passPlain)
   }

   ticket[12] <- paste(input$CampoNotasTICKETS, ticketNotaAdicional, sep = "\n")
   ticket[13] <- "avisos"
   ticket[16] <- if(input$ticketUbicacion1 == "Sin especificar") "" else
        input$ticketUbicacion1
   write.csv(ticket, file="ticketFIN.csv", na = "", row.names = F)

   Nombre <- str_split(firstname, " ")[[1]][1]
   if (is.na(str_split(firstname, " ")[[1]][2]) || str_split(firstname, " ")[[1]][2]
        == "") {
                  Apellido <- "Sin-Apellido"
                } else if (is.na(str_split(firstname, " ")[[1]][3])) {
                  Apellido <- str_split(firstname, " ")[[1]][2]
                } else {
                  Apellido <- paste(str_split(firstname, " ")[[1]][2],
        str_split(firstname, " ")[[1]][3])
                }
                Apellido <- paste(Apellido, input$client1)

                if (is.na(usuario_elegido$CMAIL1) || usuario_elegido$CMAIL1 == "") {
                  Email <- paste0("Sin-Email-", numeroClienteSinExtension)
                } else {
                  Email <- usuario_elegido$CMAIL1
                }

                system(sprintf("python tickets.py '%s' '%s' '%s' '%s' '%s'",
        Nombre, Apellido, Email, input$client1, phone))
```

```
        library("RPostgreSQL")
        drv <- dbDriver("PostgreSQL")
        con <- dbConnect(drv, user="whd", password="whd", dbname="whd",
               host="averias.emartinez.es", port=20293)
        query <- dbGetQuery(con, "SELECT client_id, job_ticket_id FROM job_ticket
               ORDER BY report_date DESC")
        dbDisconnect(con)

        query <- query[query[,1] == input$client1,]
        query <- query[complete.cases(query),]
        numeroDeTicket <- query[1,2]

        output$numeroTICKET <- renderText(paste0("Número de ticket: ",
       numeroDeTicket))
}
```

El archivo csv que leemos es el siguiente:

```
Ticket Number,*Open Date,1st Response Date,Due Date,Close Date,Status Type,Client
(user name),Localizaci?n,*Request Type (semicolon delimited),Priority
Type,Subject,*Request Detail,Tech Username,Recurso
Numbers,Notes,Room,Department,Tecnico asignado,Delete? (Y/N),NOTE: * = Field
required for new records.
,,,,,,,,,,,,,,,,,,
```

Es, como el mismo nombre indica, un fichero separado por comas (Comma-Separated Values). Los campos con asterisco son obligatorios, como la fecha del ticket, el tipo de petición y el detalle de la petición. Nosotros en nuestro programa tendremos que rellenar la siguiente fila, en donde vemos muchas comas juntas.

Leeremos este archivo con un argumento de codificación, ya que este archivo está en formato ASCII y al importarlo nos daba muchos errores, culpa de la codificación. Así ahora que tenemos cargado en memoria ese fichero, procederemos a cargar cada uno de los datos invidualmente, con tal de formar el ticket.

Insertaremos la fecha del día en que lo hagamos, en un formato predefinido usando la función *format()*, también el cliente. Luego llamaremos a la misma base de datos de clientes para obtener las ids de las localizaciones, ya que trabajan por ids, no por nombre.

Lo siguiente que introducimos es si el ticket es de nueva instalación o de infraestructura, para ello miraremos el contenido del widget. Después insertamos una nota que obtendremos primeramente del widget del *textarea*, y luego un texto adicional para saber el ususario PPPoE y contraseña, así como usuario HotSpot y contraseña.

En el siguiente campo indicamos a qué tipo de ticket se debe asignar. En nuestro caso, al ser instalaciones, siempre se asignarán a averías. Por último, introduciremos en el ticket la ubicación específica de la instalación, y guardaremos el ticket a disco, para su posterior uso en el script en Python.

Ahora seguiremos con la parte en la que debemos crear el usuario en la base de datos, ya que, para crear un ticket, primero necesitamos crear al usuario en la base de datos de tickets. Cogeremos los datos de la base de datos de clientes, y por ejemplo, Nombre, lo cogeremos a partir del campo *firstname.* Partiremos en dos trozos con el primer espacio que se encuentre como separación, y nos quedaremos con la primera parte del string, que siempre será el nombre del cliente.

La manera en que cogeremos el campo Apellido es parecida, sólo que algo más enrevesado porque a veces no existe el campo apellido o sólo aparece uno o aparecen los dos apellidos.

También cogeremos el campo Email de la base de datos de clientes. Si no existe, crearemos un usuario Sin Email seguido del número de cliente, ya que la base de datos no permite tener un mismo correo para diferentes clientes de la base de datos de tickets.

Por fin, llamaremos al script en Python vía llamada al sistema, pasándole como atributos los 5 que vemos escritos allí. Antes de explicar el script en Python seguiremos con la última parte del programa, que es la encargada de recoger el número de ticket de la base de datos, ya que al finalizar el script no hay forma sencilla de buscarlo. Lo que haremos será buscar el cliente de la tabla que nos devuelva ya ordenada, y coger la primera aparición de un número de ticket. Esto lo hacemos así porque un cliente puede tener muchos tickets abiertos, y por tanto, los ordenamos por fecha y cogemos el último.

Procedamos a explicar ahora el script en Python al que llamamos para insertar el nuevo cliente y el nuevo ticket, que ocuparán las dos siguientes páginas.

Lo más destacable de este script es que usa Selenium, que es un webscraper para Java, pero que también han hecho un *port* a Python con el que se puede trabajar tan bien como en Java, su lenguaje original. Sirve para navegar por páginas web de una manera autónoma, sin necesidad de supervisión, y puede funcionar como complemento en Firefox, pero también en PhantomJS. Este último es el que usaremos, ya que no contiene una interfaz, simplemente ejecuta el código JavaScript y ejecuta acciones, y por eso es más rápido que Firefox para navegar por la web.

Para generar el código que hemos puesto como código más abajo, hemos grabado primero una serie de acciones con un módulo de Firefox de Selenium, para luego modificarlo y exportarlo a código Python. Aun así, la mayor parte del código de la función principal ha sido reescrito.

Comenzaremos comentando por el final, ya que será por donde nuestro script empiece a ejecutar. Vemosque en el último bloque, hay un *if __name__ == "__main__"*. Aquí es donde empezará nuestro código, el resto de bloques son funciones que llamará el script o funciones para manejar errores.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoAlertPresentException
import unittest, time, re
from selenium.webdriver.common.action_chains import ActionChains
import sys

class SeleniumTICKETS(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.PhantomJS() # Ha funcionado con un apt-get install
phantomjs
        self.driver.set_window_size(1120, 550)
        self.driver.implicitly_wait(30)
        self.base_url = "http://XX.XX.XX.XX:8081/"
        self.verificationErrors = []
        self.accept_next_alert = True

    def test_selenium_t_i_c_k_e_t_s(self):
                driver = self.driver
                driver.get(self.base_url + "/helpdesk/WebObjects/Helpdesk.woa/wa")

                #Login
                driver.find_element_by_id("userName").clear()
                driver.find_element_by_id("userName").send_keys("user")
                driver.find_element_by_id("password").clear()
                driver.find_element_by_id("password").send_keys("pass")
                driver.find_element_by_css_selector("div.aquaSquareMiddle").click()

                #Nuevo cliente
                driver.find_element_by_xpath("//img[@alt='Clientes']").click()
                driver.find_element_by_css_selector("div.squareButtonMiddle").click()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.13.0
.0").clear()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.13.0
.0").send_keys(self.NAME) #NOMBRE

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.17.0
.0").clear()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.17.0
.0").send_keys(self.SURN) #APELLIDO

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.21.1
.1.0.0").clear()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.21.1
.1.0.0").send_keys(self.EMAIL) #EMAIL

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.33.0
.0").clear()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.33.0
.0").send_keys(self.CC) #CÓDIGO CLIENTE
```

```
driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.41.0.0").cle
ar()

        driver.find_element_by_name("7.25.0.0.0.2.5.5.1.8.1.3.0.1.3.1.1.0.0.1.1.41.0
.0").send_keys(self.TLFN) #TELÉFONO
                driver.find_element_by_css_selector("div.aquaMiddleSel").click()

                #Nuevo ticket
                driver.find_element_by_xpath("//img[@alt='Configuración']").click()
                element = driver.find_element_by_xpath("//div[@id='preferences-
menu']/div/div[23]")
                hover = ActionChains(driver).move_to_element(element)
                hover.perform()
                time.sleep(1)
                driver.find_element_by_xpath("//div[@id='preferences-
menu']/div/div[24]/ul/li[5]/a/div/div[2]").click()
                driver.find_element_by_xpath("//input[@name='Field Separator' and
@value='1']").click()

        driver.find_element_by_xpath("//input[@type='file']").send_keys("ticketFIN.c
sv")
                driver.find_element_by_css_selector("div.aquaMiddleSel").click()
                time.sleep(4)
                driver.find_element_by_id("logoutLink").click()

    def is_element_present(self, how, what):
        try: self.driver.find_element(by=how, value=what)
        except NoSuchElementException, e: return False
        return True

    def is_alert_present(self):
        try: self.driver.switch_to_alert()
        except NoAlertPresentException, e: return False
        return True

    def close_alert_and_get_its_text(self):
        try:
            alert = self.driver.switch_to_alert()
            alert_text = alert.text
            if self.accept_next_alert:
                alert.accept()
            else:
                alert.dismiss()
            return alert_text
        finally: self.accept_next_alert = True

    def tearDown(self):
        self.driver.quit()
        self.assertEqual([], self.verificationErrors)

if __name__ == "__main__":
        if len(sys.argv) > 4:
                SeleniumTICKETS.TLFN = sys.argv.pop()
                SeleniumTICKETS.CC = sys.argv.pop()
                SeleniumTICKETS.EMAIL = sys.argv.pop()
                SeleniumTICKETS.SURN = sys.argv.pop()
                SeleniumTICKETS.NAME = sys.argv.pop()
                unittest.main()
```

Primero, observamos que, para ejecutar el código, se deben pasar 5 o más argumentos cuando llamemos a la función. Si se cumple, Haremos un *sys.argv.pop()* para cada uno de los argumentos, tras lo cual, llamaremos a la función principal.

Primero hará un *setUp()*, que se encuentra dentro de la definición de clase de SeleniumTICKETS. En este setup, creamos un WebDriver de PhantomJS, estableciendo también su tamaño, así como ciertos parámetros como el tiempo de espera máximo, la URI principal a la que se deberá de conectar…

A continuación nos encontramos con la función de *test_selenium_t_i_c_k_e_t_s()*, en la que haremos toda la parte de *webscraping*. Primero navegaremos hacia la página web indicada como la principal de la base de datos de tickets.

Tras esto, haremos una búsqueda por ID de los elementos del Login, y los borraremos antes de introducir nada. Introduciremos el usuario y la contraseña, y luego haremos click al botón buscándolo mediante el selector de css, ya que es el único que contiene el atributo *div.aquaSquareMiddle*.

Una vez dentro de la página, haremos una búsqueda XML mediante XPath, para encontrar la imagen del cliente, y hacer click. Después de eso, se abrirá otra página y haremos click a “Nuevo cliente”.

Tras esto, nos abrirá otra página donde introducir los datos del cliente. La búsqueda de esos elementos se hace mediante id del nombre del div contenedor (por eso los nombres son tan largos). Iremos introduciendo los argumentos que le hemos pasado como parámetros.

Tras terminar, de nuevo le damos al botón de guardar. Y ahora, una vez creado el cliente, procederemos a subirle el archivo csv de ticket.

Para ello, debemos acceder a la imagen de Configuración, y dentro de él, a preferencias. Es de notar ahora, que el link que queremos pulsar dentro de preferencias está desactivado. ¿Por qué? Porque debemos hacer un *hover* sobre el elemento div “preferencias”. Esto se consigue con una función llamada *ActionChains()*. Esperamos a que se abra el desplegable un segundo, tras lo cual ya podremos pulsar los elementos seleccionados.

Una vez dentro de la importación de tickets por medio del csv, enviamos el archivo de tickets. Le damos a enviar, y esperamos manualmente cuatro segundos a que finalice la importación, tras lo cual nos deslogueamos.

En el siguiente bloque de código, explicaremos la creación de los archivos de configuración Tenda W308R.

```
if (input$select == "Sí"){
   print("Creando el archivo de configuración”)

   incProgress(5/6, detail = "Creando archivo")

   if(input$selectRouter == "Tenda W308R"){

      tryCatch({

                archivo <<- scan("data/archivo-base-tendaW308R.cfg", what =
character(), skip = 1, sep = "\n")
                ids <- vector(length = 9)
                ids[1] <- pmatch('tftp_reboot_flag=', archivo)
                ids[2] <- pmatch('wan0_pppoe_passwd=', archivo) # PPPoE Password
                ids[3] <- pmatch('wan0_pppoe_username=', archivo) # PPPoE User
                ids[4] <- pmatch('wl0_ssid=', archivo) # SSID
                ids[5] <- pmatch('wl_ssid=', archivo) # SSID
                ids[6] <- pmatch('wl_wpa_psk=', archivo) # WPA PASSWORD
                ids[7] <- pmatch('wl0_wpa_psk=', archivo) # WPA PASSWORD
                ids[8] <- pmatch('wl0_akm=', archivo) # WPA/WPA2 TYPE
                ids[9] <- pmatch('wl_akm=', archivo) # WPA/WPA2 TYPE

                MACapp <- paste0(toupper(limpiaMAC(input$MAC_1)), '.cfg')
                #Iniciales wifi
                iniciales <- 0
                for (k in 1:length(str_split(firstname, " ")[[1]])){
                  iniciales[k] <- str_split(str_split(firstname, " ")[[1]],
"")[[k]][1]
                }
                iniciales <- paste0(iniciales, collapse = '')

                archivo[ids[1]] <- sprintf("tftp_reboot_flag=%s", runif(1,0,1)) #
Random number para tiempo de reconexión a tftp
                archivo[ids[2]] <- sprintf("wan0_pppoe_passwd=%s", passPlain)
                archivo[ids[3]] <- sprintf("wan0_pppoe_username=%s", numeroCliente)
                archivo[ids[4]] <- sprintf("wl0_ssid=EMARTINEZ_%s", iniciales)
                archivo[ids[5]] <- sprintf("wl_ssid=EMARTINEZ_%s", iniciales)
                archivo[ids[6]] <- sprintf("wl_wpa_psk=emartinez%s",
numeroClienteSinExtension)
                archivo[ids[7]] <- sprintf("wl0_wpa_psk=emartinez%s",
numeroClienteSinExtension)
                archivo[ids[8]] <- sprintf("wl0_akm=%s", "psk2")
                archivo[ids[9]] <- sprintf("wl_akm=%s", "psk2")

                write(archivo, paste0("./archivos-samba/", MACapp))
                system(sprintf("smbclient -U user //XX.XX.XX.XX/tftpSRV t1l2cm3r -c
\"put %s %s\"", sprintf("./archivos-samba/%s", MACapp), MACapp))

                output$erroresMAC <- renderText("Correcto. No han habido errores
subiendo el archivo de configuración")

              }, warning = function(w){
                output$erroresMAC <- renderText("Error. Han habido errores subiendo
el archivo de configuración")
              })
            }
```

Lo primero que hacemos tras superar el primer *if*, es incrementar la bara de progreso. Este *if* abarcará los tres tipos de router. El primer *if* es para el router Tenda W308R. Lo primero que hacemos es envolver todo en un elemento *tryCatch()* por si hay algún error leyendo, procesando, o escribiendo a disco el archivo.

Para leer el archivo de un router Tenda, hemos preparado uno especial, que es simplemente el archivo de configuración de un router configurado de fábrica. Leeremos con *scan()* el archivo, saltándonos la primera línea de comentario. Es de apreciar que el comando de asignación que usamos es el de *scope* global. Esto es así ya que tenemos que hacer esa variable visible globalmente, y por eso la asignamos de esa manera. Así estará disponible, por ejemplo, en el widget de descarga directa del archivo de configuración.

Básicamente, lo que hacemos es buscar con la función *pmatch()* los índices donde se encuentran los string puestos como argumento, para luego cambiarlos. Los parámetros a modificar son, por ejemplo, el *tftp_reboot_flag*, que sirve para que cuando la luz vuelva tras un corte de luz masivo, no todos los router se conecten a la vez al servidor TFTP, ya que sólo puede gestionar una petición simultáneamente.

Entre otros, también tenemos la opción de configurar el usuario PPPoE y su contraseña, así como la SSID WiFi, la contraseña de la conexión wifi, y un par de parámetros para configurar que la seguridad del wifi sea de WPA2-PSK.

Tras modificar estos parámetros, lo escribimos a disco,con su MAC en mayúscula y su extensión *.cfg* . Luego lo que haremos será subirlo a un servicio compartido SAMBA, en la que el almacenamiento no será problema. Para ello, usamos el comando *smbclient* de Linux, con usuario, contraseña, dirección de la carpeta compartida y el comando *put*. Más adelante se explicará cómo se ha montado el almacenamiento compartido y cómo hemos hecho para que la dirección de almacenamiento del TFTP server sea la carpeta compartida.

Si ha ido todo bien, se lanzará un mensaje correcto a la aplicación, mientras que si algo ha fallado, saldrá un mensaje de error.

Ahora hablaremos de la creación de los archivos XML de los Grandstream HT502. Estos archivos son bastante extensos y sus tags tienen la forma de la letra P seguido de un número. Para saber qué significan cada uno de sus parámetros, tendremos que navegar hasta la página web del Grandstream HT502. Allí encontraremos una lista extensiva de los parámetros con su funcionalidad [42].

En la misma página web tenemos un generador de XML, pero preferimos usar R para esta generación, ya que es más sencilla, más práctica, y más rápida.

El código lo encontraremos en las tres páginas siguientes:

```
else if (input$selectRouter == "Grandstream HT502"){

tryCatch({

archivo <- xmlParse("data/archivo-base-grandstreamHT502.xml")

posicion <- vector(length=34)
posicion[1] <- xpathSApply(archivo, "//P8")
xmlValue(posicion[[1]]) <- "2"

posicion[2] <- xpathSApply(archivo, "//P30")
xmlValue(posicion[[2]]) <- "185.44.24.10"

posicion[3] <- xpathSApply(archivo, "//P38")
xmlValue(posicion[[3]]) <- "48"

posicion[4] <- xpathSApply(archivo, "//P57")
xmlValue(posicion[[4]]) <- "8"

posicion[5] <- xpathSApply(archivo, "//P64")
xmlValue(posicion[[5]]) <- "CET-1CEST-2,M3.5.0/02:00:00,M10.5.0/03:00:00"

posicion[6] <- xpathSApply(archivo, "//P109")
xmlValue(posicion[[6]]) <- "0"

posicion[7] <- xpathSApply(archivo, "//P133")
xmlValue(posicion[[7]]) <- "0"

posicion[8] <- xpathSApply(archivo, "//P189")
xmlValue(posicion[[8]]) <- "1"

posicion[9] <- xpathSApply(archivo, "//P190")
xmlValue(posicion[[9]]) <- "1"

posicion[10] <- xpathSApply(archivo, "//P231")
xmlValue(posicion[[10]]) <- "1"

posicion[11] <- xpathSApply(archivo, "//P243")
xmlValue(posicion[[11]]) <- "1"

posicion[12] <- xpathSApply(archivo, "//P246")
xmlValue(posicion[[12]]) <- "GMT+0BST,M3.5.0,M10.5.0"

posicion[13] <- xpathSApply(archivo, "//P258")
xmlValue(posicion[[13]]) <- "1"

posicion[14] <- xpathSApply(archivo, "//P277")
xmlValue(posicion[[14]]) <- "1"

posicion[15] <- xpathSApply(archivo, "//P401")
xmlValue(posicion[[15]]) <- "0"

posicion[16] <- xpathSApply(archivo, "//P854")
xmlValue(posicion[[16]]) <- "10"

posicion[17] <- xpathSApply(archivo, "//P901")
xmlValue(posicion[[17]]) <- "XXXX"
```

```
posicion[18] <- xpathSApply(archivo, "//P2363")
xmlValue(posicion[[18]]) <- "0"

posicion[19] <- xpathSApply(archivo, "//P4200")
xmlValue(posicion[[19]]) <- "{[6789]xxxxxxxx| x+ | *x+ | *xx*x+ }"

posicion[20] <- xpathSApply(archivo, "//P4567")
xmlValue(posicion[[20]]) <- "1"

posicion[21] <- xpathSApply(archivo, "//P5001")
xmlValue(posicion[[21]]) <- "0"

posicion[22] <- xpathSApply(archivo, "//P26003")
xmlValue(posicion[[22]]) <- "0"

posicion[23] <- xpathSApply(archivo, "//mac")
xmlValue(posicion[[23]]) <- toupper(limpiaMAC(input$MAC_1))

posicion[24] <- xpathSApply(archivo, "//P58")
xmlValue(posicion[[24]]) <- "8"

posicion[25] <- xpathSApply(archivo, "//P59")
xmlValue(posicion[[25]]) <- "8"

posicion[26] <- xpathSApply(archivo, "//P60")
xmlValue(posicion[[26]]) <- "8"

posicion[27] <- xpathSApply(archivo, "//P61")
xmlValue(posicion[[27]]) <- "8"

posicion[28] <- xpathSApply(archivo, "//P62")
xmlValue(posicion[[28]]) <- "8"

posicion[29] <- xpathSApply(archivo, "//P46")
xmlValue(posicion[[29]]) <- "8"

posicion[30] <- xpathSApply(archivo, "//P98")
xmlValue(posicion[[30]]) <- "8"

posicion[31] <- xpathSApply(archivo, "//P92")
xmlValue(posicion[[31]]) <- "XX"

posicion[32] <- xpathSApply(archivo, "//P93")
xmlValue(posicion[[32]]) <- "XX"

posicion[33] <- xpathSApply(archivo, "//P94")
xmlValue(posicion[[33]]) <- "XX"

posicion[34] <- xpathSApply(archivo, "//P95")
xmlValue(posicion[[34]]) <- "XX"

MAC_GS <<- sprintf("cfg%s.xml", tolower(limpiaMAC(input$MAC_1)))

# Elegimos la IP según si es servidor de empresas o de clientes
ipSIP <- if(input$selectParticularEmpresa == "Particular") "XX.XX.XX.XX" else
"XX.XX.XX.XX"

# Password LARGA del SIP
passwordLargaSIP <- input$passSIP
```

```
# Password del PPPoE telefonía
passwordPPPoETLFN <- passPlain

#Añadimos siblings
addSibling(posicion[[1]], newXMLNode("P196", "XXX"))
addSibling(posicion[[1]], newXMLNode("P83", passwordPPPoETLFN))
addSibling(posicion[[1]], newXMLNode("P2", "XXX"))
addSibling(posicion[[1]], newXMLNode("P35", input$telefono))
addSibling(posicion[[1]], newXMLNode("P47", ipSIP))
addSibling(posicion[[1]], newXMLNode("P82", numeroCliente))
addSibling(posicion[[1]], newXMLNode("P269", "XXX"))
addSibling(posicion[[1]], newXMLNode("P34", passwordLargaSIP))
addSibling(posicion[[1]], newXMLNode("P3", firstname))

saveXML(archivo, file=paste0("./archivos-samba-http/", MAC_GS), prefix = "")
system(sprintf("smbclient -U user //XX.XX.XX.XX/httpSRV pass -c \"put %s %s\"",
sprintf("./archivos-samba-http/%s", MAC_GS), MAC_GS))

}, warning = function(w){
output$erroresMAC <- renderText("Error. Han habido errores subiendo el archivo de
configuración")
})
}
```

Tras la comprobación *if-else*, pondremos un *tryCatch()*. Tras esto, usaremos el paquete XML para trabajar con el archivo XML del teléfono en cuestión. Parsearemos el archivo con la función *xmlParse()*, tras lo cual iremos usando recursivamente las funciones *xpathSApply()* y *xmlValue()* para ir buscando y asignando valores, respectivamente.

Después de esto, usaremos también la función *addSibling()* para añadir un nuevo nodo XML, en el que pondremos nuevos datos de parámetros. Para crear estos nodos XML usaremos la función *newXMLNode()*.

Para finalizar, guardaremos el archivo XML con la función especial *saveXML()*, tras lo cual haremos una llamada al sistema vía *smbclient*.

Pasamos ahora a la parte del envío de comandos para el router Tenda W300D mediante la API que dispone GenieACS.

Al igual que siempre, comprobamos con un *if* tras lo cual insertamos un bloque *tryCatch()*. Primero jugaremos con la MAC, tras lo cual borraremos cualquier preset anterior que pudiera tener, ya que un mismo router puede servir para varios clientes a lo largo de su vida útil.

Calculamos las iniciales del cliente, y vamos a hacer que el CPE haga un *inform* cada 15 segundos.

Ahora nos encontramos con la parte más engorrosa de todo esto: generar un texto en JSON para luego insertarlo por medio de HTTP a GenieACS

```
else if (input$selectRouter == "Tenda W300D"){

tryCatch({

deviceID <- tolower(limpiaMAC(input$MAC_1))
system(sprintf("curl -i 'http://185.44.24.18:7557/presets/%s' -X DELETE", deviceID))

iniciales <- 0
for (k in 1:length(str_split(firstname, " ")[[1]])){
     iniciales[k] <- str_split(str_split(firstname, " ")[[1]], "")[[k]][1]
}
iniciales <- paste0(iniciales, collapse = '')

informInterval <- "15"

       datos1 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.SSID\", \"value\":
\"EMARTINEZ_%s\" }", iniciales)
       datos2 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.X_BROADCOM_COM_WlanAdapter.W
lVirtIntfCfg.1.WlSsid\", \"value\": \"EMARTINEZ_%s\" }", iniciales)
       datos3 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.X_BROADCOM_COM_WlanAdapter.W
lVirtIntfCfg.1.WlWpaPsk\", \"value\": \"XXX%s\" }", XXX)
       datos4 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.X_BROADCOM_COM_WlanAdapter.W
lVirtIntfCfg.2.WlWpaPsk\", \"value\": \"XXX%s\" }", XXX)
       datos5 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.X_BROADCOM_COM_WlanAdapter.W
lVirtIntfCfg.3.WlWpaPsk\", \"value\": \"XXX%s\" }", XXX)
       datos6 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.X_BROADCOM_COM_WlanAdapter.W
lVirtIntfCfg.4.WlWpaPsk\", \"value\": \"XXX%s\" }", XXX)
       datos7 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.PreSharedKey.1.KeyPassphrase
\", \"value\": \"XXX%s\" }", numeroClienteSinExtension)
       datos8 <- "{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.LANDevice.1.WLANConfiguration.1.BeaconType\", \"value\":
\"basic\" }"
       datos9 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.ManagementServer.PeriodicInformInterval\", \"value\": \"%s\"
}", informInterval)
       datos10 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.X_Tenda_QuicksetupCfgObject.EasyInstUser\", \"value\":
\"%s\" }", numeroCliente)
       datos11 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.X_Tenda_QuicksetupCfgObject.EasyInstPwd\", \"value\": \"%s\"
}", passPlain)
       datos12 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.3.WANConnectionDevice.1.WANPPPConnection.6.Usernam
e\", \"value\": \"%s\" }", numeroCliente)
       datos13 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.3.WANConnectionDevice.1.WANPPPConnection.6.Passwor
d\", \"value\": \"%s\" }", passPlain)
       datos14 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.3.WANPPPConnection.1.Usernam
e\", \"value\": \"%s\" }", numeroCliente)
```

```
        datos15 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.3.WANPPPConnection.1.Passwor
d\", \"value\": \"%s\" }", passPlain)
        datos16 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.1.WANPPPConnection.1.Usernam
e\", \"value\": \"%s\" }", numeroCliente)
        datos17 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.1.WANPPPConnection.1.Passwor
d\", \"value\": \"%s\" }", passPlain)
        datos18 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.4.WANPPPConnection.1.Usernam
e\", \"value\": \"%s\" }", numeroCliente)
        datos19 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.1.WANConnectionDevice.4.WANPPPConnection.1.Passwor
d\", \"value\": \"%s\" }", passPlain)
        datos20 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.3.WANConnectionDevice.1.WANPPPConnection.1.Usernam
e\", \"value\": \"%s\" }", numeroCliente)
        datos21 <- sprintf("{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.WANDevice.3.WANConnectionDevice.1.WANPPPConnection.1.Passwor
d\", \"value\": \"%s\" }", passPlain)
        datos22 <- sprintf("{ \"type\": \"add_tag\", \"tag\": \"%s\" }", deviceID)
        datos23 <- "{ \"type\": \"value\", \"name\":
\"InternetGatewayDevice.Time.Enable\", \"value\": \"true\" }"

        datosFin <- sprintf("'{ \"weight\": 30, \"precondition\":
\"{\\\"summary.serialNumber\\\":\\\"%s\\\"}\", \"configurations\": [ %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s ] }'",
deviceID, datos1, datos2, datos3, datos4, datos5, datos6, datos7, datos8, datos9,
datos10, datos11, datos12, datos13, datos14, datos15, datos16, datos17, datos18,
datos19, datos20, datos21, datos22, datos23)
         system(sprintf("curl -i 'http://XX.XX.XX.XX:7557/presets/%s' -X PUT --data
%s", deviceID, datosFin))
              })
            }
          }
        }) # End barra de progreso
          enable("sincronizar")
    } # End IF(campo client1 != vacío)
})
```

Iremos concatenando strings y haciendo un uso intensivo de la función *sprintf()* del paquete *stringr*. Tendremos que enviarle, para cada dato que queramos asignar como *preset*, el tipo, el nombre y el valor de cada asignación del preset. Aquí configuraremos el usuario PPPoE, su contraseña, configuraciones del WiFi… etc.

Por último ensamblaremos todos los string datos para luego hacer una llamada al sistema vía *curl*. Así automáticamente crearemos nuestro preset.

Para finalizar, vemos que la ejecución termina totalmente, volviendo a activar el botón de sincronización, cerrando todas las llaves.

Ya sólo nos queda explicar el lado del servidor de las dos tabs anexas al programa de Router Express, que sirven para la creación de los archivos de configuración. Pondremos todo el código y comentaremos los temas más relevantes:

```
    output$client <- renderText({
      paste("Código de cliente: ", input$client)
    })
    output$initials <- renderText({
      paste("Iniciales del cliente: ", input$initials)
    })
    output$user <- renderText({
      paste("Usuario PPPoE: ", input$user)
    })
    output$pass <- renderText({
      paste("Contraseña PPPoE: ", input$pass)
    })
    output$mac<- renderText({
      paste("MAC del Router: ", toupper(limpiaMAC(input$mac)))
    })
    # Botón de descarga, lo que se hace tras la pulsación es esto:
    output$downloadData <- downloadHandler(
      filename = function() {
        MAC
      },
      content = function(file) {
        write(archivo, file)
      }
    )

    observeEvent(
      input$upload,{
        archivo <<- scan("data/archivo-base-tendaW308R.cfg", what = character(),
skip = 1, sep = "\n")
        ids <<- vector(length = 7)
        ids[1] <<- pmatch('tftp_reboot_flag=', archivo) # Parámetro de reconexión
        ids[2] <<- pmatch('wan0_pppoe_passwd=', archivo) # PPPoE Password
        ids[3] <<- pmatch('wan0_pppoe_username=', archivo) # PPPoE User
        ids[4] <<- pmatch('wl0_ssid=', archivo) # SSID
        ids[5] <<- pmatch('wl_ssid=', archivo) # SSID
        ids[6] <<- pmatch('wl_wpa_psk=', archivo) # WPA PASSWORD
        ids[7] <<- pmatch('wl0_wpa_psk=', archivo) # WPA PASSWORD
        ids[8] <<- pmatch('wl0_akm=', archivo) # WPA/WPA2 TYPE
        ids[9] <<- pmatch('wl_akm=', archivo) # WPA/WPA2 TYPE

        MAC <<- paste0(toupper(limpiaMAC(input$mac)), '.cfg')
        archivo[ids[1]] <<- sprintf("tftp_reboot_flag=%s", runif(1,0,1))
        archivo[ids[2]] <<- sprintf("wan0_pppoe_passwd=%s", input$pass)
        archivo[ids[3]] <<- sprintf("wan0_pppoe_username=%s", input$user)
        archivo[ids[4]] <<- sprintf("wl0_ssid=EMARTINEZ_%s", input$initials)
        archivo[ids[5]] <<- sprintf("wl_ssid=EMARTINEZ_%s", input$initials)
        archivo[ids[6]] <<- sprintf("wl_wpa_psk=XXX%s", XXX)
        archivo[ids[7]] <<- sprintf("wl0_wpa_psk=XXX%s", XXX)
        tryCatch({
          write(archivo, paste0("./archivos-samba/", MAC))
          system(sprintf("smbclient -U user //XX.XX.XX.XX/tftpSRV pass -c \"put %s
%s\"", sprintf("./archivos-samba/%s", MAC), MAC))
          output$ok <- renderText("0 Errores")
          print(Sys.time())
        }, warning = function(w) {
          output$ok <- renderText("Han habido errores en la conexión TFTP")
        })
      }
    )
```

Este es la tab completa de la creción de los archivos MAC de los Tenda W308R. Es prácticamente idéntica a lo que hay implementado en la aplicación principal, así que no merece la pena explicarla.

Pasemos ahora a la tab de la creación del archivo de configuración para el GrandStream HT502:

```
    output$nombreGS <- renderText({
      paste("Nombre del cliente: ", input$nombreGS)
    })
    output$userGS <- renderText({
      paste("Usuario PPPoE: ", input$userGS)
    })
    output$passGS <- renderText({
      paste("Contraseña PPPoE: ", input$passGS)
    })
    output$telefonoGS <- renderText({
      paste("Número de teléfono: ", input$telefonoGS)
    })
    output$passSIPGS <- renderText({
      paste("Constraseña SIP: ", input$passSIPGS)
    })
    output$passSIPGS <- renderText({
      paste("Constraseña SIP: ", input$passSIPGS)
    })
    output$selectParticularEmpresaGS <- renderText({
      paste("Particular o Empresa: ", input$selectParticularEmpresaGS)
    })
    output$macGS <- renderText({
      paste("MAC: ", tolower(limpiaMAC(input$macGS)))
    })
    output$downloadDataGS <- downloadHandler(
      filename = function() {
        MACGS
      },
      content = function(file) {
        write(archivo, file)
      }
    )

observeEvent(
      input$uploadGS,{

        archivo <<- scan("data/archivo-base-grandstreamHT502.xml", what =
character(), sep = "\n")

        ids <<- vector(length=24)
        ids[1] <<- pmatch('\t\t<P8>', archivo)
        ids[2] <<- pmatch('\t\t<P30>', archivo) # NTP
        ids[4] <<- pmatch('\t\t<P38>', archivo)
        ids[5] <<- pmatch('\t\t<P57>', archivo)
        ids[6] <<- pmatch('\t\t<P64>', archivo) # Time Zone
        ids[8] <<- pmatch('\t\t<P109>', archivo)
        ids[9] <<- pmatch('\t\t<P133>', archivo)
        ids[10] <<- pmatch('\t\t<P189>', archivo)
        ids[11] <<- pmatch('\t\t<P190>', archivo)
        ids[12] <<- pmatch('\t\t<P231>', archivo)
```

```
        ids[13] <<- pmatch('\t\t<P243>', archivo)
        ids[14] <<- pmatch('\t\t<P246>', archivo) # Time Zone
        ids[15] <<- pmatch('\t\t<P258>', archivo)
        ids[16] <<- pmatch('\t\t<P277>', archivo)
        ids[17] <<- pmatch('\t\t<P401>', archivo)
        ids[18] <<- pmatch('\t\t<P854>', archivo)
        ids[19] <<- pmatch('\t\t<P901>', archivo) # Puerto 8080
        ids[20] <<- pmatch('\t\t<P2363>', archivo) # ANTES NO ESTABA ABAJO
        ids[21] <<- pmatch('\t\t<P4200>', archivo)
        ids[22] <<- pmatch('\t\t<P4567>', archivo)
        ids[23] <<- pmatch('\t\t<P5001>', archivo)
        ids[24] <<- pmatch('\t\t<P26003>', archivo) # ANTES NO ESTABA ABAJO
        ids[25] <<- pmatch('<mac>', archivo)
        ids[26] <<- pmatch('\t\t<P58>', archivo) # Vocoder 2
        ids[27] <<- pmatch('\t\t<P59>', archivo) # Vocoder 3
        ids[28] <<- pmatch('\t\t<P60>', archivo) # Vocoder 4
        ids[29] <<- pmatch('\t\t<P61>', archivo) # Vocoder 5
        ids[30] <<- pmatch('\t\t<P62>', archivo) # Vocoder 6
        ids[31] <<- pmatch('\t\t<P46>', archivo) # Vocoder 7
        ids[32] <<- pmatch('\t\t<P98>', archivo) # Vocoder 8
        ids[33] <<- pmatch('\t\t<P92>', archivo) # DNS
        ids[34] <<- pmatch('\t\t<P93>', archivo) # DNS
        ids[35] <<- pmatch('\t\t<P94>', archivo) # DNS
        ids[36] <<- pmatch('\t\t<P95>', archivo) # DNS

        MACGS <<- sprintf("cfg%s.xml", tolower(limpiaMAC(input$macGS)))

        # Elegimos la IP según si es servidor de empresas o de clientes
        ipSIP <<- if(input$selectParticularEmpresaGS == "Particular") "XX.XX.XX.XX"
else "XX.XX.XX.XX"

        # Password LARGA del SIP
        passwordLargaSIP <<- input$passSIPGS

        # Password del PPPoE telefonía
        passwordPPPoETLFN <<- input$passGS

        # Hacemos hueco al final y añadimos los P-valores que no existían en el
archivo de configuración
        l <<- length(archivo)

        archivo[l+9] <<- archivo[l]
        archivo[l+8] <<- archivo[l-1]
        # Añadimos las líneas inexistentes en la parte final del archivo
        archivo[l+7] <<- "\t\t<P196>XXX</P196>"
        archivo[l+6] <<- sprintf("\t\t<P83>%s</P83>", passwordPPPoETLFN)
        archivo[l+5] <<- "\t\t<P2>XXX</P2>"
        archivo[l+4] <<- sprintf("\t\t<P35>%s</P35>", input$telefonoGS)
        archivo[l+3] <<- sprintf("\t\t<P47>%s</P47>", ipSIP) # Puede ser de clientes
o empresas

        extension <<- tail(unlist(strsplit(input$userGS, "e")), n=1)
        if (!(extension == "t")) numeroClienteGS <<- paste0(input$userGS, "t")
        else numeroClienteGS <<- input$userGS
        archivo[l+2] <<- sprintf("\t\t<P82>%s</P82>", numeroClienteGS)
```

```
        archivo[l+1] <<- "\t\t<P269>XXX</P269>"
        archivo[l] <<- sprintf("\t\t<P34>%s</P34>", passwordLargaSIP)
        archivo[l-1] <<- sprintf("\t\t<P3>%s</P3>", input$nombreGS)

        # Añadimos el resto
        archivo[ids[1]] <<- "\t\t<P8>2</P8>" # PPPoE Option
        archivo[ids[2]] <<- "\t\t<P30>XXX</P30>"
        archivo[ids[4]] <<- "\t\t<P38>48</P38>"
        archivo[ids[5]] <<- "\t\t<P57>8</P57>"
        archivo[ids[6]] <<- "\t\t<P64>CET-1CEST-
2,M3.5.0/02:00:00,M10.5.0/03:00:00</P64>"
        archivo[ids[8]] <<- "\t\t<P109>0</P109>"
        archivo[ids[9]] <<- "\t\t<P133>0</P133>"
        archivo[ids[10]] <<- "\t\t<P189>1</P189>"
        archivo[ids[11]] <<- "\t\t<P190>1</P190>"
        archivo[ids[12]] <<- "\t\t<P231>1</P231>"
        archivo[ids[13]] <<- "\t\t<P243>1</P243>"
        archivo[ids[14]] <<- "\t\t<P246>GMT+0BST,M3.5.0,M10.5.0</P246>"
        archivo[ids[15]] <<- "\t\t<P258>1</P258>"
        archivo[ids[16]] <<- "\t\t<P277>1</P277>"
        archivo[ids[17]] <<- "\t\t<P401>0</P401>"
        archivo[ids[18]] <<- "\t\t<P854>10</P854>"
        archivo[ids[19]] <<- "\t\t<P901>XXX</P901>"
        archivo[ids[20]] <<- "\t\t<P2363>0</P2363>"
        archivo[ids[21]] <<- "\t\t<P4200>{[6789]xxxxxxxx| x+ | *x+ | *xx*x+
}</P4200>"
        archivo[ids[22]] <<- "\t\t<P4567>1</P4567>"
        archivo[ids[23]] <<- "\t\t<P5001>0</P5001>"
        archivo[ids[24]] <<- "\t\t<P26003>0</P26003>"
        archivo[ids[25]] <<- sprintf("<mac>%s</mac>",
toupper(limpiaMAC(input$macGS)))
        archivo[ids[26]] <<- "\t\t<P58>8</P58>"
        archivo[ids[27]] <<- "\t\t<P59>8</P59>"
        archivo[ids[28]] <<- "\t\t<P60>8</P60>"
        archivo[ids[29]] <<- "\t\t<P61>8</P61>"
        archivo[ids[30]] <<- "\t\t<P62>8</P62>"
        archivo[ids[31]] <<- "\t\t<P46>8</P46>"
        archivo[ids[32]] <<- "\t\t<P98>8</P98>"
        archivo[ids[33]] <<- "\t\t<P92>XXX</P92>"
        archivo[ids[34]] <<- "\t\t<P93>XXX</P93>"
        archivo[ids[35]] <<- "\t\t<P94>XXX</P94>"
        archivo[ids[36]] <<- "\t\t<P95>XXX</P95>"

        tryCatch({
          write(archivo, paste0("./archivos-samba-http/", MAC_GS))
          system(sprintf("smbclient -U user //XX.XX.XX.XX/httpSRV pass -c \"put %s
%s\"", sprintf("./archivos-samba-http/%s", MAC_GS), MAC_GS))
          output$okGS <- renderText("0 Errores")
          print(Sys.time())
        }, warning = function(w) {
          output$ok <- renderText("Han habido errores en la creación del archivo")
        })
      }
    )

})
```

Aquí notamos rápidamente que la metodología usada no ha sido la misma que con la del programa principal. Esta fue la primera versión que se hizo sin usar el paquete XML, debido a su complejidad inicial de uso.

La idea fue leer el archivo por medio de *scan()*, y hacer lo que hicimos en el archivo de configuración del Tenda W308R: Buscar los índices con *pmatch()* y luego cambiarlos con el valor deseado. El problema aquí, a parte de la lentitud, fue que para cada valor que quisiéramos, teníamos que poner obligatoriamente tabuladores, tal como exige el formato XML. También, para alojar valores nuevos, teníamos que extender el archivo y poner las últimas dos líneas al final, siendo un poco caótico el código.

Y hasta aquí el código referente al archivo server.R.

4.3. Servicio de DHCP Option 66

DHCP66 es un servicio que el CPE tiene que tener implementado en software. En nuestro caso, como ya dijimos anteriormente, tenemos los routers Grandstream HT502 y Tenda W308R. Hemos necesitado de un servidor TFTP y otro HTTP para enviarles archivos, así como hemos tenido que configurar los routers MikroTik para que ofrecieran esa opción.

Procederemos primero a explicar la configuración de los servidores TFTP y HTTP, luego hablaremos sobre cómo hemos montado un servicio SAMBA para cada servidor. A continuación, explicaremos lo que hemos hecho para activar los servicios en cada MikroTik, y por último explicaremos cómo hemos hecho para redirigir unas consultas DNS del equipo Grandstream.

4.3.1. Configuración de los servidores TFTP y HTTP

Pasaremos ahora a hablar acerca de los servidores de archivos que hemos implantado en el mismo servidor Ubuntu 14.04 LTS.

Apache

Prácticamente no hemos tenido que realizar ninguna configuración. Descargamos el paquete del repositorio oficial de Ubuntu, y pusimos todos los archivos dentro de la carpeta */var/www/html/*.

Xinetd

Descargamos los archivos del repositorio oficial de Ubuntu *xinetd*, *tftp* y *tftpd*. Tras esto configuramos el archivo en */etc/xinetd.d/tftp* de esta manera:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /media/tftpSRV
disable         = no
}
```

Donde lo más destacable es que montamos un servidor Samba en */media/tftpSRV*.

4.3.2. Configuración del servidor Samba

Lo que queremos hacer es montar un almacenamiento virtual en nuestra máquina, es decir, crear una carpeta en nuestro servidor que cuando la llamemos, nos traiga los archivos desde otro servicio de almacenamiento más grande y más seguro [43].

Usaremos un sisema de archivos virtual CIFs, que ya viene con el sistema operativo, pero instalaremos samba previamente.

```
sudo apt-get update
sudo apt-get install samba
```

Creamos el directorio */media/tftpSRV*:

```
mkdir /media/tftpSRV
sudo mount -t cifs //XX.XX.XX.XX/tftpSRV /media/tftpSRV -o username=X
```

Y ya tendremos nuesto almacenamiento compartido listo. Lo mismo sucederá para nuestro servidor HTTP.

```
mkdir /media/httpSRV
sudo mount -t cifs //XX.XX.XX.XX/httpSRV /media/httpSRV -o username=X
```

4.3.3. Activación de DHCP66 en cada MikroTik

Para configurar la opción 66, tuvimos que insertar en cada MikroTik un parámetro que a continuación procederemos a explicar con detalle.

Primero accedemos a un MikroTik en cuestión. Cuando accedes vía *WinBox* a un MikroTik, podemos ver el panel de administración principal:

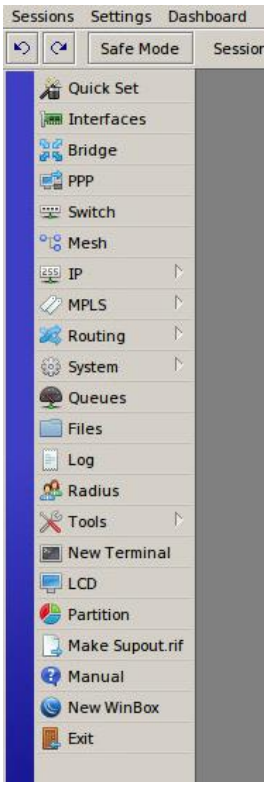

Figura 65: Vista vía WinBox de un MikroTik

En él, navegaremos hasta IP, luego a DHCP Server:

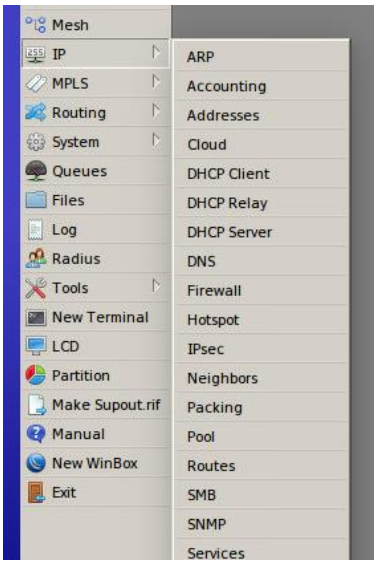

Figura 66: Seleccionando DHCP Server

Y luego, dentro de DHCP Server, seleccionamos la pestaña Options, tras lo cual le haremos click al símbolo de suma. Nos saldrá una ventana, donde pondremos los siguientes valores:

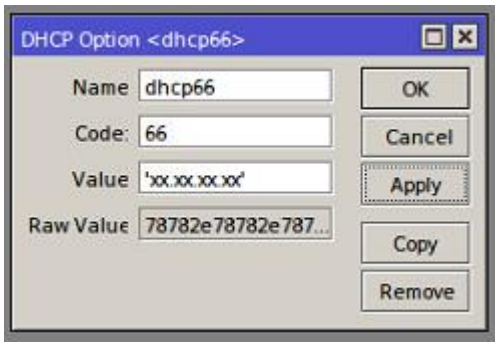

Figura 67: Creando opción dhcp66

Tras esto, guardamos, y vemos el resultado en la pestaña de Options:

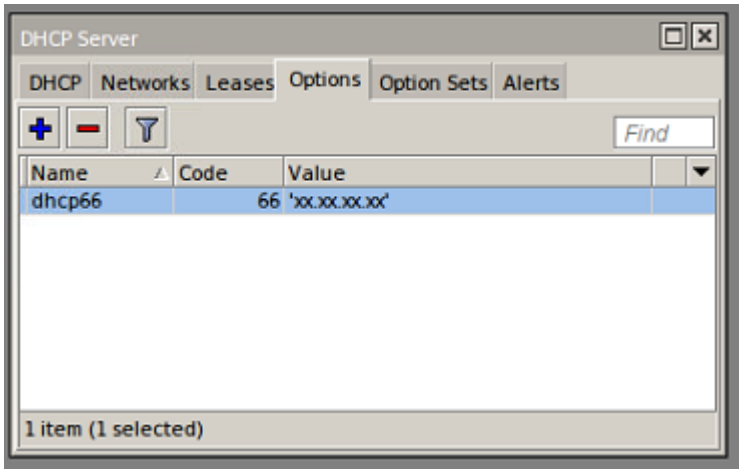

Figura 68: DHCP66 insertado

Y ya quedará configurado ese MikroTik con la opción 66. Ahora cuando gestione una sesión DHCP, al terminar de asignar una IP a un CPE, en la respuesta final, le enviará esta opción.

4.3.4. Script para cambio de DNS masivo en cada MikroTik

Nos dimos cuenta que en los routers GrandStream venía por defecto el servidor de autoconfiguración, con lo que hacía una especie de DHCP66 pero sin ningún servidor al que preguntar.

De fábrica, vienen configurados a que busquen su archivo de configuración a *fm.grandstream.com*, vía HTTP al puerto 80. Así que lo que hicimos fue crear un servidor HTTP en nuestro servidor. También hicimos redirigir en las DNS del servidor DNS de la empresa ese nombre a la dirección de nuestro servidor HTTP.

Para ello, hicimos un script en *bash* que se conectaba a cada MikroTik, y le enviaba el comando para establecer su tabla DNS como hemos dicho anteriormente.

Para extraer las IPs, nos conectamos al programa *The Dude*, con el que se gestiona toda la red, propiedad de MikroTik. Allí sacamos la lista de routers MikroTik en formato csv.

Tras haber sacado las IPs, las incorporamos a nuestro script en bash. Lo escribimos tal que así:

```
#!/bin/bash

for i in XX.XX.XX.XX XX.XX.XX.XX ...
do
        sh scriptMKT.sh $i
done
```

Vemos un bucle *for* en el que recorremos las IP y vamos llamando a otro script en bash. Este script es el siguiente:

```
#!/bin/bash

echo $1
ssh pass -p pass ssh user@$1 -t 'ip dhcp-server network set
numbers=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
dns-server=XX.XX.XX.XX'
```

Con lo que ahora, para su tabla DNS, en todos sus campos, sólo existe un servidor DNS principal al que preguntar. En este servidor DNS principal simplemente asociamos la dirección *fm.grandstream.com* a la dirección de nuestra máquina, y ya está todo resuelto para la autoconfiguración de los Grandstream.

Capítulo 5. Conclusiones y trabajos futuros.

En el presente trabajo fin de grado se ha realizado una aplicación diseñada a medida para la empresa, apoyándose en los protocolos DHCP66 y TR069. El objetivo perseguido era ahorrar tiempo al dar servicio al nuevo cliente, pudiendo dedicarlo a otros trabajos menos rutinarios, aumentando la satisfacción del cliente al ver que se le atiende mucho mejor. Ahora en la práctica, incluso un técnico puede llevarse un router sin configurar pues no habrá problema, ya que desde consola se usa Router Express y el router queda automáticamente configurado.

Como conclusiones, hemos podido adentrarnos un poco más al terreno de las grandes operadoras del mercado, en cuanto a la autoconfiguración de sus dispositivos y los servicios de replicación y sincronización de datos. Hoy en día es indispensable recurrir a estos servicios si se quiere ahorrar tiempo y dinero en una empresa a la que cada día lleguen más y más clientes.

Hemos profundizado bastante en el lenguaje de programación R, así como en el uso de sus paquetes. Hemos aprendido también un poco de Python, muy útil allá donde vayas, la “navaja suiza” de la programación. Hemos aprendido mucho en el uso de sistemas Linux, específicamente de Ubuntu, que es donde hemos alojado todos los servicios de los que hemos estado hablando a lo largo del trabajo. Además, hemos manejado muchos parámetros tanto de routers cliente como de routers de tráfico.

Como trabajos futuros, se espera que esta aplicación sea la controladora de todas las bases de datos, así como también en un futuro más lejano, que se cree una base de datos asociada, y que de él pendan todos los clientes nuevos.

Así, sería una especie de CRM en la que todas las bases de datos penderían de ella. Cuando un nuevo cliente llegue, no se añadirá a la base de datos de clientes, sino que directamente vendrán a la aplicación web en Shiny y darán de alta al cliente desde allí.

Si esto llega a ser así, se valorará escribir el código de la nueva aplicación servidora en Java o Python, dependiendo de las necesidades futuras de la empresa. También se valorará qué base de datos SQL será la más apropiada, quizá una MySQL o PostgreSQL.

Como detalles a mejorar de la aplicación actual, en un futuro próximo se establecerá la creación de un teléfono en la centralita de voz. Eligiendo en la aplicación si es un nuevo número o portabilidad, la aplicación haría las subsecuentes gestiones para dar de alta el nuevo número en la centralita de voz. También quizá se valore el hecho de insertar un ticket vía conexión a base de datos en vez de insertar los datos vía webscraping, ya que es muy costosa en tiempo y en computación.

Bibliografía.

- [1] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html
- [2] https://kb.iu.edu/d/adov
- [3] http://linux.die.net/man/8/xinetd
- [4] http://tftpd32.jounin.net/
- [5] http://kin.klever.net/pumpkin
- [6] http://news.netcraft.com/archives/2015/01/15/january-2015-web-server-survey.html
- [7] http://httpd.apache.org/
- [8] http://nginx.org/
- [9] https://www.broadband-forum.org/cwmp.php
- [10] https://genieacs.com/
- [11] https://github.com/zaidka/genieacs
- [12] https://github.com/akcoder/genieacs
- [13] http://www.easycwmp.org/
- [14] http://www.axiros.com/products-portfolio/management-via-tr-069-ssh-telenet-snmp-axess-gdm
- [15] http://www.avsystem.com/products/unified-device-management-platform/
- [16] https://cran.r-project.org/
- [17] https://www.r-project.org/about.html
- [18] https://www.python.org/
- [19] https://www.java.com/es/
- [20] http://pythoniza.me/analisis-java-vs-python/
- [21] https://es.wikipedia.org/wiki/Markdown
- [22] http://www.informit.com/library/content.aspx?b=STY_Sql_24hours&seqNum=9
- [23] https://www.mysql.com/
- [24] http://www.postgresql.org/
- [25] https://www.sqlite.org/
- [26] http://www.couchbase.com/nosql-resources/what-is-no-sql
- [27] https://www.mongodb.org/
- [28] http://www.cisco.com/
- [29] http://www.mikrotik.com/
- [30] http://www.juniper.net/us/en/
- [31] http://www.w3.org/XML/
- [32] http://www-01.ibm.com/support/knowledgecenter/SS6RBX_11.4.3/com.ibm.sa.xml.design.doc/topics/c_history.html
- [33] https://www.rstudio.com/
- [34] https://cran.r-project.org/web/packages/data.table/index.html
- [35] http://shiny.rstudio.com/
- [36] http://shiny.rstudio.com/tutorial/lesson1/
- [37] http://www.forowifi.com/forum/showthread.php?t=36381
- [38] https://github.com/zaidka/genieacs/wiki/API-Reference
- [39] http://rmarkdown.rstudio.com/
- [40] http://bootswatch.com/
- [41] http://fontawesome.io/icons/
- [42] http://www.grandstream.com/support/resources/?title=HandyTone%20502
- [43] https://help.ubuntu.com/community/Samba/SambaClientGuide
- [44] https://www.rstudio.com/products/RStudio/

ANEXO A: Instalación RStudio y gestión de paquetes

Para instalar RStudio, primeramente tendremos que descargar e instalar el núcleo de RStudio, es decir, R.

Para ello, navegaremos a la página oficial de R CRAN y nos bajaremos la versión para nuestro dispositivo.

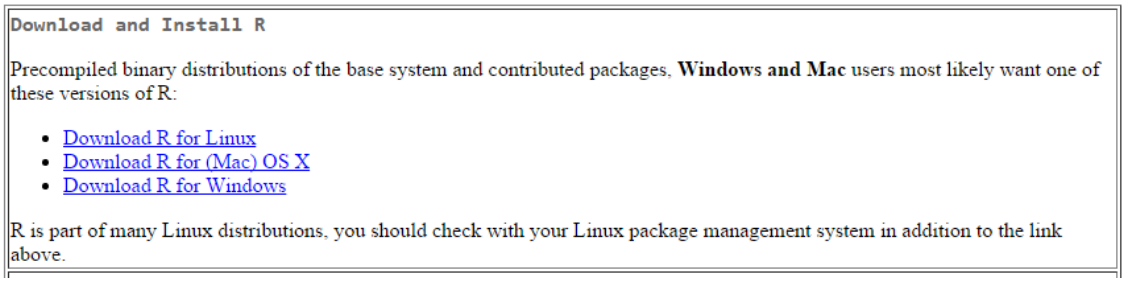

Figura 69: Página web de R CRAN

Le haremos click, y nos llevará a una siguiente página, dependiendo del sistema operativo. Esta, por ejemplo, es a la que nos lleva si hacemos click en "Windows":

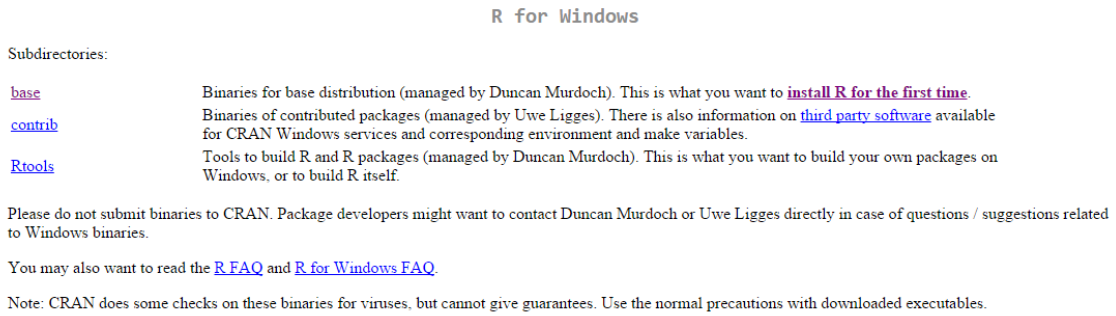

Figura 70: Página de descarga para Windows

Las opciones de descarga son *base*, *contrib* y *Rtools.* Lo que queremos es núcleo de R, así que descargaremos *base*. *Contrib* son binarios contribuidos de paquetes, por si queremos extender R con más paquetes, según la gestión de Uwe Ligges. Por último, *Rtools* son herramientas para crear paquetes, más orientado a desarrolladores de R.

Tras pulsar a *base*, se nos abrirá la siguiente página, donde finalmente podremos descargar R:

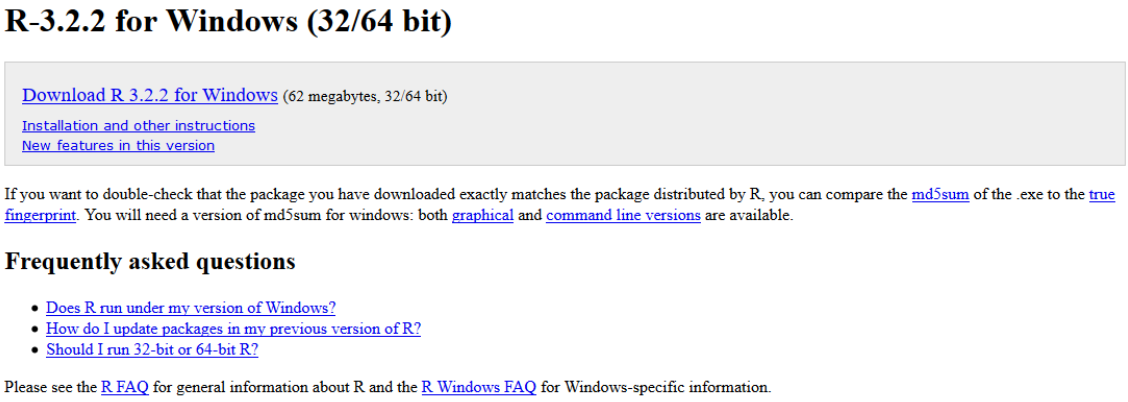

Figura 71: Finalmente, podemos descargar R

Haremos click a “Download R X.X.X for Windows”.

Tras descargar el archivo, lo abrimos, y nos guiará mediante un asistente de instalación.

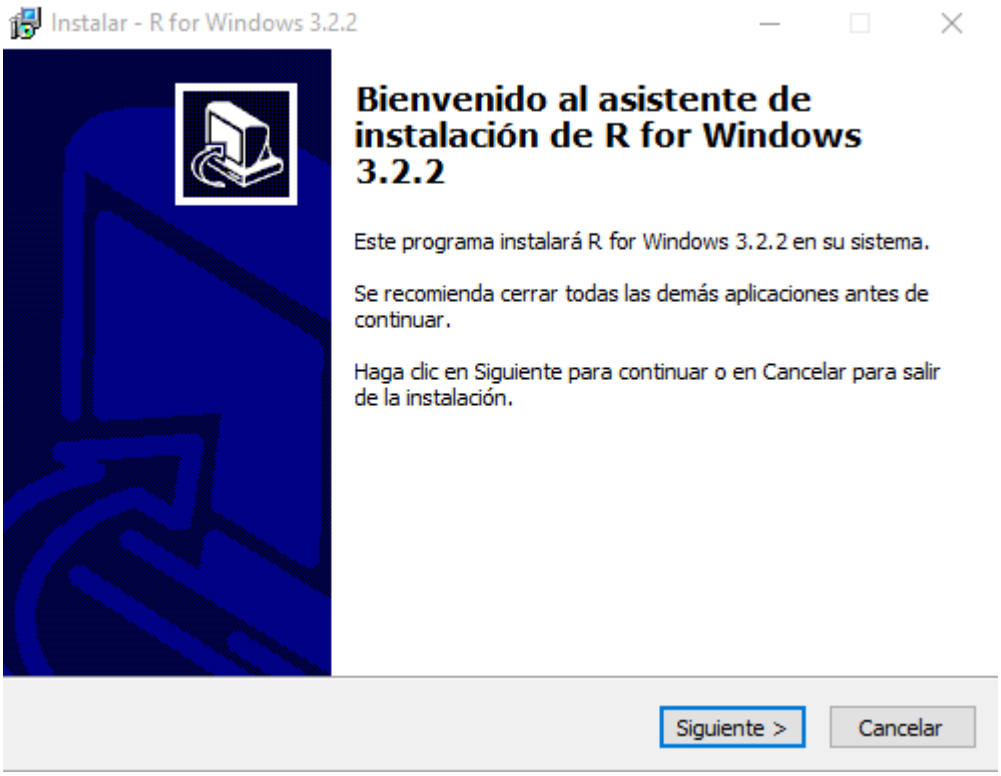

Figura 72: Pantalla principal del asistente de instalación para Windows

Tras aceptar los términos de uso, nos dará a escoger la carpeta de instalación. Es mejor dejarla así por defecto. Luego nos da la opción de escoger si instalar los *Core Files*, los archivos de 32-bit, de 64-bit o la traducción de mensajes de eror/advertencia:

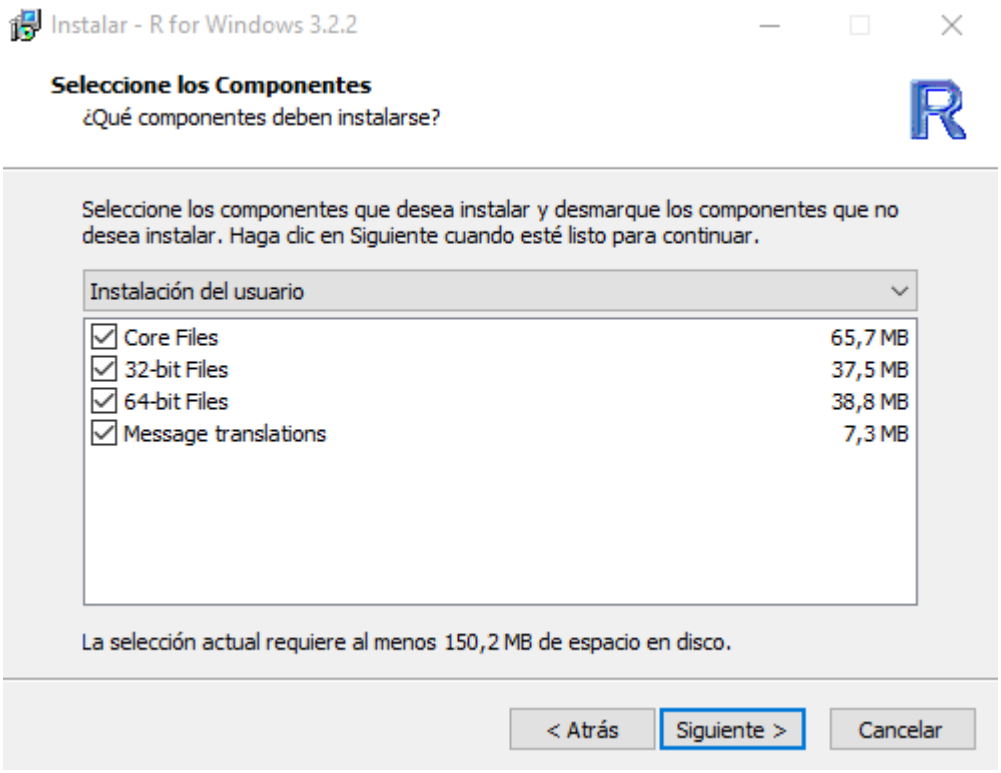

Figura 73: Elección de componentes a instalar

Dejaremos todo marcado como nos viene por defecto. Aun así, es conveniente, muchas veces, hacer búsquedas de errores en inglés. En inglés hay más documentación, por lo que sería una opción interesante el descartar la instalación de “Message translations”.

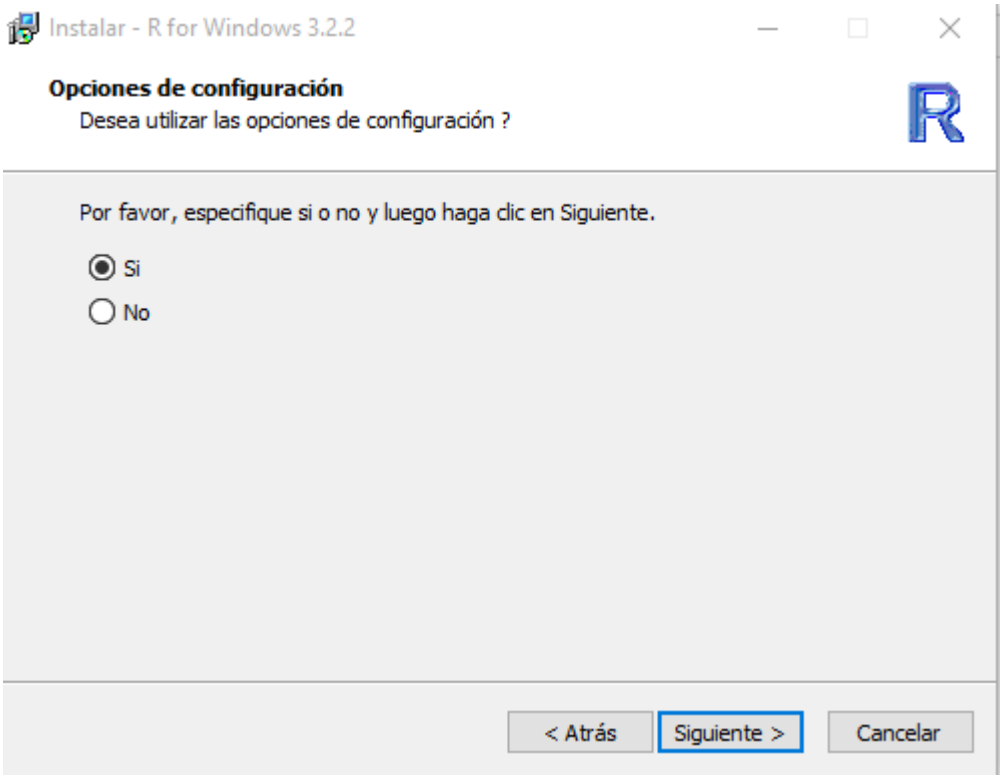

Figura 74: Elección de opciones de configuración

Escogeremos “Sí” a las opciones de configuración. En la siguiente opción, nos dará a elegir entre MDI o SDI. En principio escogeremos MDI si queremos que R se ejecute de modo ventana única, y SDI si queremos que cada elemento pueda navegar libremente por la pantalla

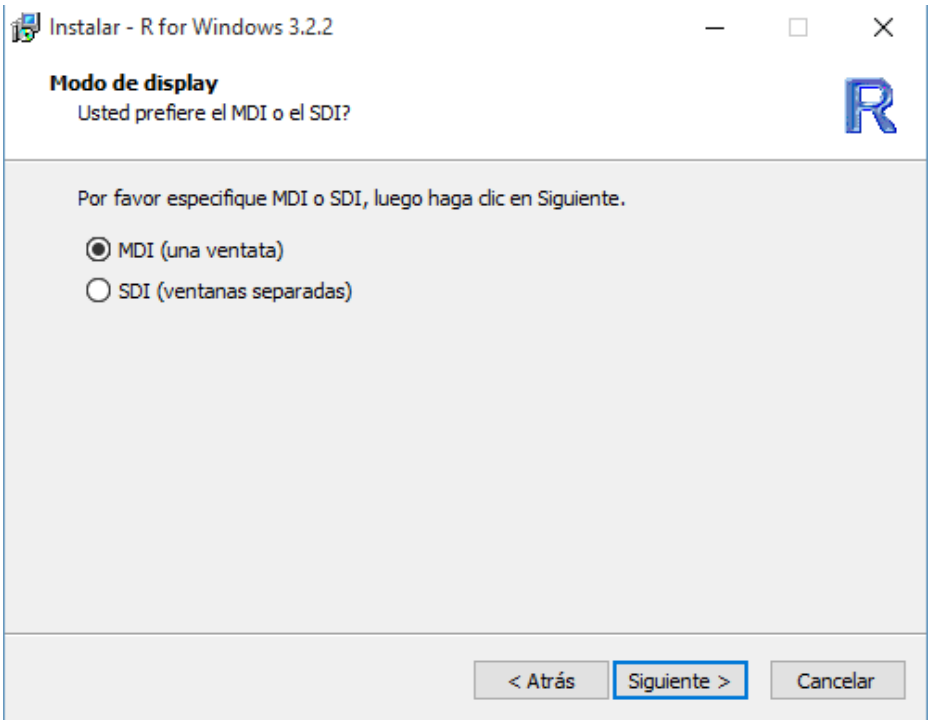

Figura 75: MDI o SDI

Luego nos preguntará acerca de cómo preferimos la ayuda, si en HTML o en texto plano. Elegimos HTML, aunque hay pocas diferencias apreciables. Le daremos a siguiente en la siguiente pestaña también, tras lo cual llegaremos a esta ventana:

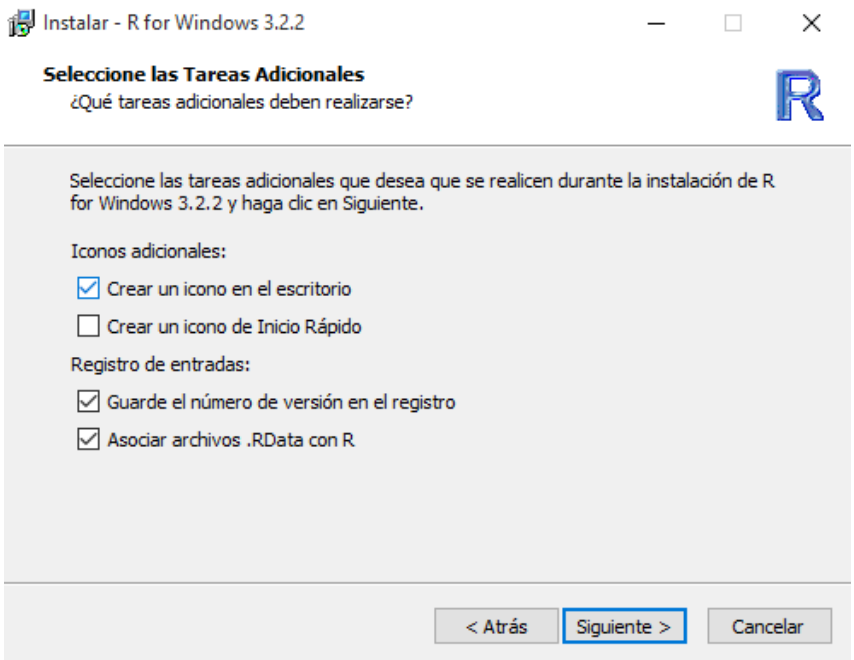

Figura 76: Tareas adicionales

En la que escogeremos lo más apropiado para nosotros, si crear un icono en el Escritorio o en Inicio Rápido. Por defecto también queremos que se guarde el número de versión de R en el registro de Windows y que Windows asocie la extensión .RData con R. La instalación seguirá hasta completar. Ya tenemos instalado R.

Procedamos a la instalación de RStudio.

Navegaremos hasta la página web de RStudio, donde allí le seleccionaremosa "Instalar RStudio". Tras ello, se nos abrirá una página en la que podremos elegir la versión que deseemos [44]. Tendremos la opción entre descargar Rstudio Desktop y RStudio Server.

Descargaremos RStudio Desktop, para trabajar localmente de forma cómoda.

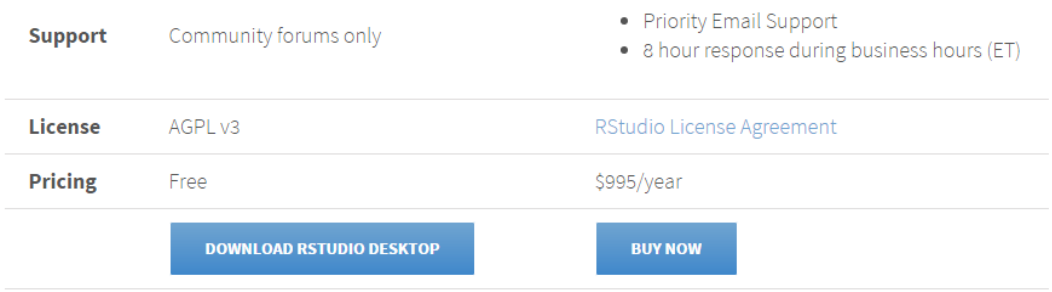

Figura 77: Zona de descarga de RStudio

Tras ello, se nos abrirá una segunda página donde podremos elegir nuestra versión para el sistema operativo que tengamos instalado. En nuestro caso, será Windows.

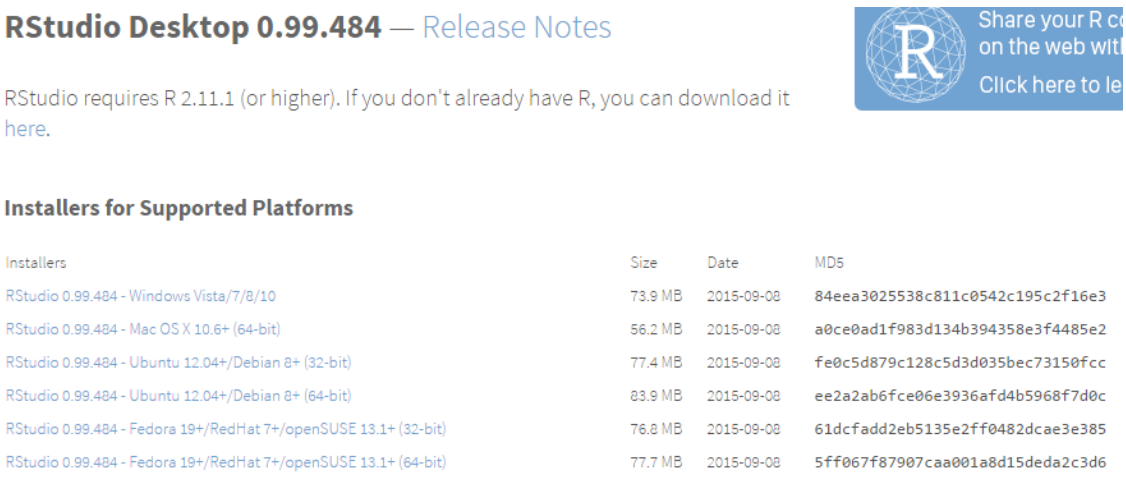

Figura 78: Descarga de RStudio Desktop en función del sistema operativo instalado

Después de descargar nuestro archivo, procederemos a ejecutar el instalador.

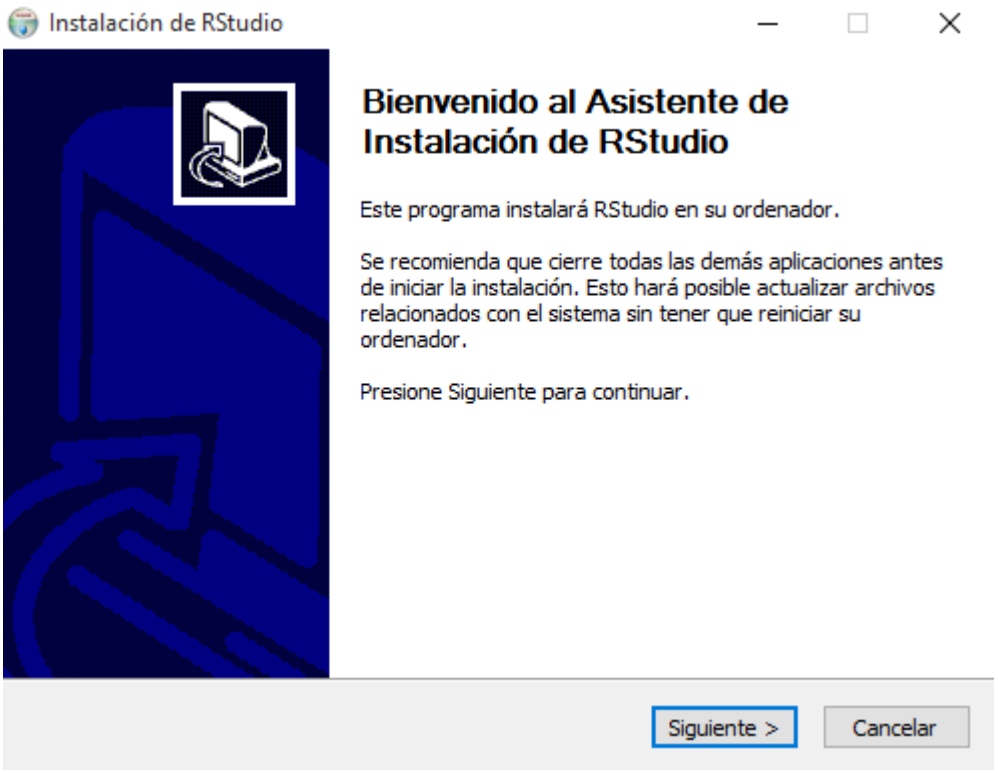

Figura 79: Pantalla principal del asistente de instalación de RStudio Desktop

En la siguiente ventana, podremos elegir la ruta de la instalación, la dejaremos por defecto. Tras darle de nuevo a siguiente, el programa automáticamente se instalará sin más configuraciones. Si queremos configurar algo en específico, ya será dentro de RStudio la configuración que pongamos.

Así, abrimos RStudio, que nos mostrará la siguiente pantalla principal:

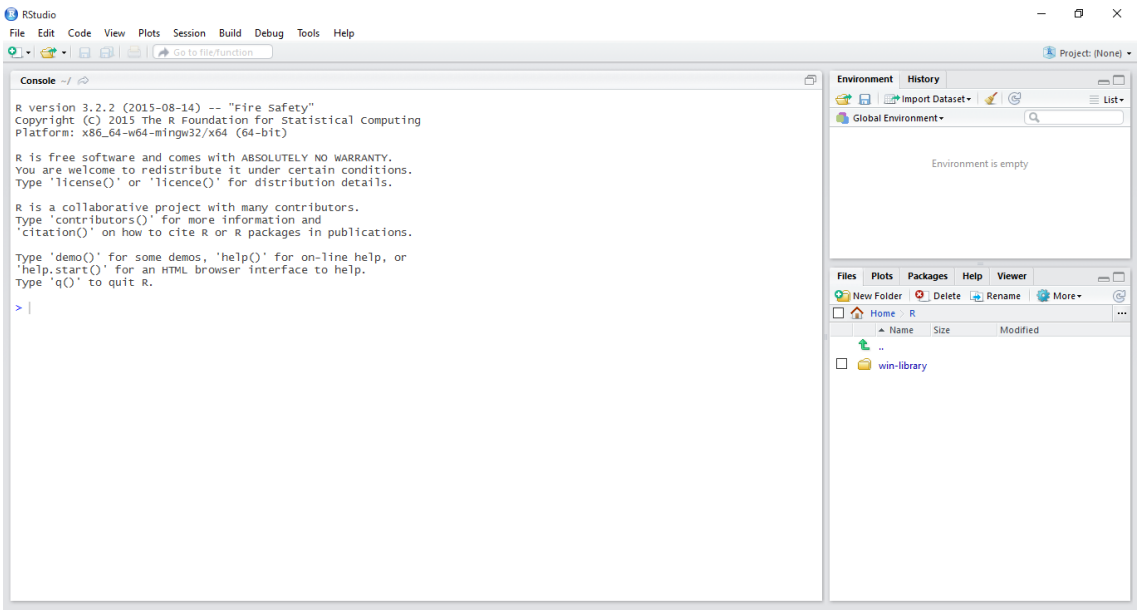

Figura 80: Pantalla principal de RStudio

Así que ya estamos listos para instalar nuevos paquetes. Imaginemos que neceistamos los paquetes *shiny*, *stringr*, *dplyr*, *gdata* y *digest* para nuestro proyecto. Procedamos a instalarlos.

El comando es *install.packages(c("shiny", "stringr", "dplyr", "gdata", "digest"))*, y aquí veremos al comando haciendo su trabajo:

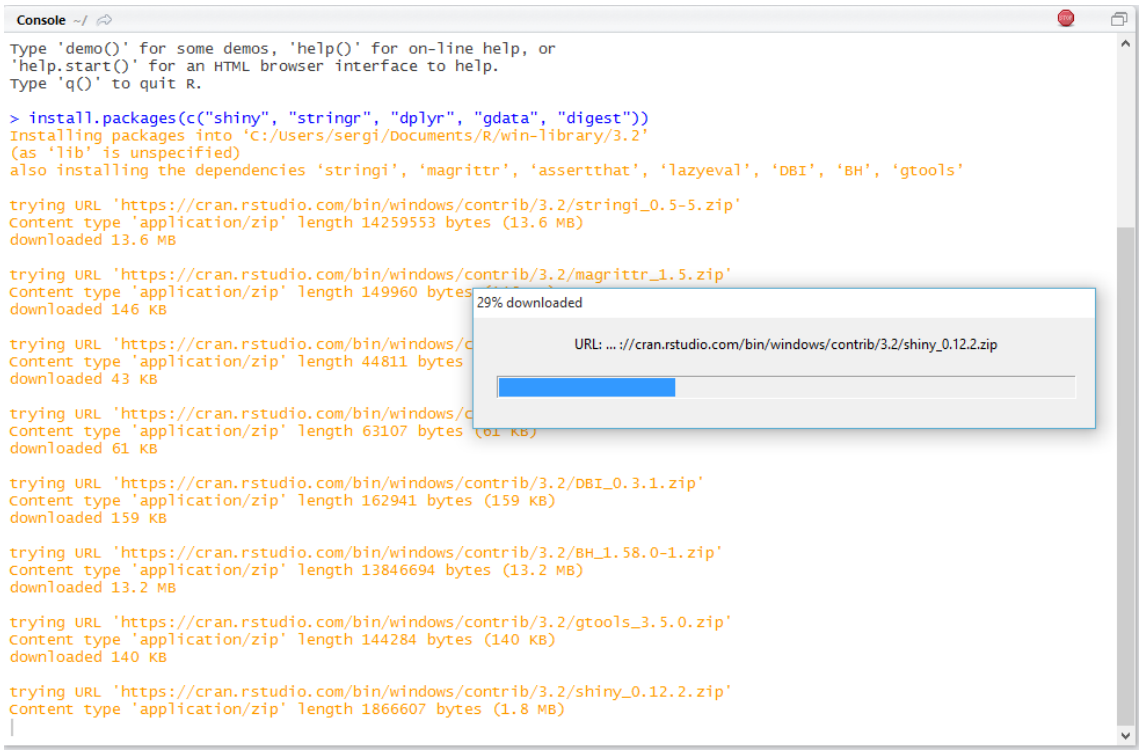

Figura 81: Instalando paquetes

Si quieremos mantener al día los paquetes, lo recomendable es usar *update.packages()*, tal como vemos en la imagen:

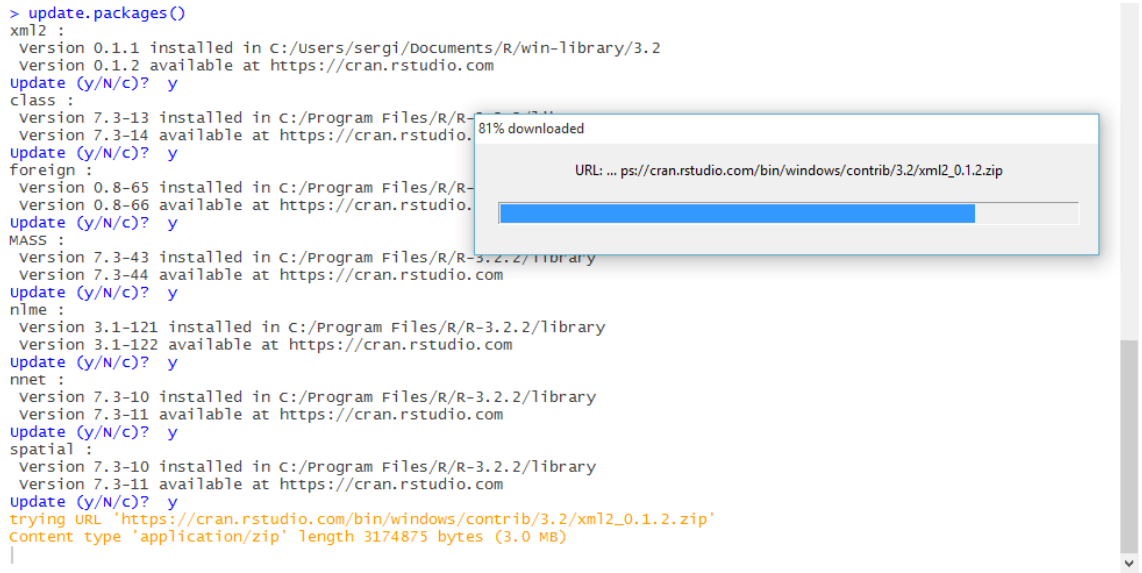

Figura 82: Actualizando paquetes