

Sensores de red movil

Información sobre su programación y su buena programación



Índice

[Introducción 2](#_Toc12113616)

[GSM: RTL-SDR 2](#_Toc12113617)

[LTE: BladeRF 2](#_Toc12113618)

[Envió de los datos 3](#_Toc12113619)

[Captura de redes GSM 3](#_Toc12113620)

[Introducción 3](#_Toc12113621)

[Explicación de los diversos subprogramas 3](#_Toc12113622)

[Programa de escaneo modificado: grgsm\_scanner 3](#_Toc12113623)

[Programa de recogida de datos: grgsm\_livemon\_headless 3](#_Toc12113624)

[Programa de envío de datos: sender.py 4](#_Toc12113625)

[Programa de reinicio del ordenador: reinicio.sh 4](#_Toc12113626)

[Programa principal: script.sh 4](#_Toc12113627)

[Metodología de instalación 4](#_Toc12113628)

[Conclusión 4](#_Toc12113629)

[Capturas de redes 4G 4](#_Toc12113630)

[Introducción 4](#_Toc12113631)

[Explicación de los diversos subprogramas 5](#_Toc12113632)

[Escaneo de las frecuencias: cell\_search 5](#_Toc12113633)

[Estudio de las frecuencias: estudio.py 5](#_Toc12113634)

[Programa de selección de frecuencias: vectordeescaneo.py y selectordefrecuencia.py 5](#_Toc12113635)

[Programa de escaneo de frecuencias: PDSCH\_UE 5](#_Toc12113636)

[Programa de descifrado de datos: descodificador.py 6](#_Toc12113637)

[Programa de envío de datos: sender.py 6](#_Toc12113638)

[Programa principal: script.sh 6](#_Toc12113639)

[Metodología de instalación 6](#_Toc12113640)

[Conclusión 7](#_Toc12113641)

[Unión de los dos sistemas 7](#_Toc12113642)

# Introducción

La introducción la voy a dividir en dos partes. En la primera voy a hablar sobre los dispositivos que usamos para captar gsm y en la segunda mitad voy a escribir sobre el dispositivo que usamos para captar LTE y redes de última generación. Por último, voy a añadir un apéndice sobre la subida de los datos a la plataforma de NASERTIC.

Cabe añadir que todos estos datos se encuentran en un repositorio de github: <https://github.com/JLashN/imsicatcher>

## GSM: RTL-SDR

Los dispositivos que usamos para captar gsm son rtl-sdr. Estos dispositivos con pequeños dispositivos similares a USBs de los cuales sale una antena. Compramos dos tipos: Unos con conexión SMA y otros con conexión coaxial (como la conexión de la televisión, por ejemplo). Estos últimos contaban con el problema de que la conexión coaxial no es el estándar para antenas y por tanto no podíamos cambiar las antenas para optimizar la recepción de gsm.

La programación de estos dispositivos no fue muy complicada debido a que todo el código estaba ya hecho y solo había que rehacer pequeñas partes o juntar diferentes programas en uno. Además el código era libre por lo que no fue difícil llegar a los repositorios de estos programas y descargarlos.

Sin embargo, fue notable la dificultad de garantizar la estabilidad de estos dispositivos. Esto se debe a que:

* **No conocemos cual es su construcción interna:** Esto no debería ser un problema ya que si la interfaz del dispositivo esta bien implementada, la implementación de este nos es irrelevante. Sin embargo, la implementación de la interfaz ha sido programa copiando otros dispositivos y no esta optimizada para el dispositivo en cuestión
* **El código de internet pese a ser libre no estaba bien documentado:** Esto no facilitaba en absoluto la optimización de estas librerías para garantizar la estabilidad.
* **Las librerías se corrompen de manera arbitraria:** Hay momentos en los que las librerías dejan de funcionar bien y el dispositivo no funciona correctamente y se queda colgado. Por lo que la única solución es reiniciar.

Tratare este aspecto más adelante y ahora voy a seguir hablando sobre la programación de LTE.

## LTE: BladeRF

El dispositivo que hemos usado principalmente para captar LTE han sido unos BladeRF. También hemos usado otros SDR como puede ser UHD-ETTUS u otros pero el que mejor resultado han dado han sido los BladeRF además de ser los más baratos.

Como he comentado antes con los gsm, el código es libre por lo que se podía editar con facilidad para obtener nuestro uso. Además, al contar los SDR con más potencia hay muchas más suites desarrolladas para obtener distintas funcionalidad como simular GPS, simular una estación base de LTE, … esta última es la que hemos usado editándola para obtener nuestro propósito de captar y descodificar tráfico LTE.

Por otra parte, al ser más potente tiene también mas problemas que el GSM para lograr un sistema automático. Así que a los problemas de antes hay que añadir:

* **Usa una FPGA para emitir/recibir señales:** Esto en si mismo no es un problema porque una FPGA es un procesador que se reprograma. El problema viene en que al introducirse la programación en la FPGA hay veces en las que no permite ser reprogramado y hay que desenchufar y volver a enchufar. **Esto es un serio problema** porque reiniciando el ordenador no se soluciona si no esta bien configurada la bios y expulsando el dispositivo desde el ordenador tampoco porque la corriente sigue llegando a la FPGA y sigue atascada. **Este es el problema en el que estamos ahora más atentos.**

Comentare en más detalle la programación de LTE más adelante.

## Envió de los datos

El envió de los datos no tiene mayor historia. Simplemente extrae los datos de lo capturado y los envía al servidor ORION, que a su vez funciona de buffer para una base de datos MONGO.

Ahora que he terminado la introducción comenzaré explicando la programación de los rtl-sdr para captar redes gsm.

# Captura de redes GSM

## Introducción

Para capturar redes gsm hemos usado varios lenguajes de programación:

* Hemos usado **Python 2.7.1** para crear el programa que envía los datos a la plataforma.
* Hemos usado **Python 2.7.1** también para editar el programa gr-gsm para obtener lo que queremos y que no nos de cosas superfluas al escanear, por ejemplo.
* Hemos usado **Bash** para programar un programa que reinicia el ordenador si se queda colgado.
* Por último, hemos usado **Bash** para programar un programa que haga que todo el proceso se realice correctamente.

Para que todos estos programas se usen correctamente hay que instalar primero una serie de programas como viene documentado dentro del manual que se encuentra en el [repositorio](https://github.com/JLashN/imsicatcher).

## Explicación de los diversos subprogramas

### Programa de escaneo modificado: grgsm\_scanner

Este programa lo que hace es escanear las frecuencias de gsm y sacarlas por pantalla. He modificado la salida para que se ajuste a los parámetros que necesito para que el programa funcione correctamente. El resto del programa es exactamente igual al de la suite de gr-gsm.

### Programa de recogida de datos: grgsm\_livemon\_headless

Este programa es exactamente el de la suite de gr-gsm. La única diferencia es que en el script he adaptado los parámetros para que funcione con 3 antenas y funcione en más rangos de frecuencia. Este programa lo que hace es monitorizar estos datos y volcarlos a un puerto de localhost.

### Programa de envío de datos: sender.py

Este programa no hace otra cosa que enviar los datos a la plataforma. Lee los datos que se han guardado por tshark en un archivo llamado file.txt, los analiza y los envía. Después de esta lectura y envió, se borra el fichero para evitar la saturación del disco con futuros escaneos.

### Programa de reinicio del ordenador: reinicio.sh

Este programa lo que hace es evitar que el ordenador se bloquee. Para ello, analiza el uso de la cpu del ordenador y el tiempo que lleva ejecutándose para saber si el ordenador se ha bloqueado y si es así, lo reinicia.

### Programa principal: script.sh

Este programa lo que hace es coordinar los otros subprogramas. Su funcionamiento es el siguiente: Cada cierto número de iteraciones fijadas de antemano, el programa escanea para encontrar frecuencias libres (ya que las frecuencias de gsm teóricamente no permanecen fijas. Sino ni siquiera se escanearían más que la primera vez) y luego escanea durante 60 segundos en esa frecuencia mientras lo guarda en un archivo.

Cuando ha terminado de escanear, el programa da paso al sender.py que envía los datos a la plataforma de Nasertic.

## Metodología de instalación

Como ya he indicado en la introducción, la metodología de instalación viene detallada en el manual que esta dentro del repositorio. Sin embargo, para automatizarlo correctamente hay que tener en cuenta una serie de cosas:

* Es necesario, en principio, un reinicio al día para evitar otros fallos del sensor. Este reinicio no tiene por que ser a una determinada hora. En los sensores, esta programado mediante el Cron a las 00:00 pero puede ser a cualquier otra hora. Los reinicios son muy rápidos (1 minuto) así que no se pierde demasiada información.
* Para iniciar el programa, es necesario llamar a arrancador.sh que esta dentro de la carpeta final del repositorio. De esta manera, se inician correctamente los dos scripts (El de 2G y el de 4G).

## Conclusión

La instalación de los nodos GSM no tienen mayor complicación. Además, la estabilidad se puede garantizar con una serie de reinicios programados así que no tiene mayor historia.

Hay una gran de subprogramas que he necesitado para diversas tareas que se coordinan correctamente mediante un programa principal.

# Capturas de redes 4G

## Introducción

Para la captura de redes 4G hemos usado diversos lenguajes y diversas técnicas. Todos los programas necesarios se encuentran dentro de la carpeta FinalLTE:

* Hemos usado **Python 2.7.1** para hacer un programa que genere un estudio de las frecuencias utilizadas para así ahorrarnos escaneos.
* Hemos usado **Python 2.7.1** para hacer un programa que descodifique los paquetes de 4G ya que no tienen un formato normal sino que están encriptados (Con una encriptación por defecto que es legal desencriptar).
* Hemos usado también **Python 2.7.1** para crear un par de programas que a partir del escaneo de frecuencias decidan cuales se va a escanear.
* Hemos usado **C++** para editar el código fuente de PDSCH\_UE que se encuentra en la carpeta srsLTE/lib/examples/ para que redirija lo que se escucha a un archivo que luego podamos enviar.
* Hemos usado **Bash** para crear el programa principal que coordine todo.
* Por último, hemos usado **Python 2.7.1** para crear el programa que envie los datos a la plataforma.

Para la correcta instalación de estos programas, hay que instalar algunos drivers y alguna librería. Lo detallaré en un apartado posterior.

La suite principal de la que sacaremos la funcionalidad de escaneo de las redes 4G será de srsLTE que es la carpeta que está dentro del [repositorio](https://github.com/JLashN/imsicatcher).

## Explicación de los diversos subprogramas

### Escaneo de las frecuencias: cell\_search

Este programa no esta editado: Es tal cual el programa que viene en la suite de srsLTE. Lo que hace es recorrer una banda (indicada al programa por parámetros) indicando cuales de las frecuencias de la banda contienen información (porque el 4G se está emitiendo ahí). El funcionamiento es el mismo que el de grgsm\_scanner pero seleccionando la banda.

Sin embargo, este programa tiene un problema: La velocidad del escaneo. Así como el escaneo de GSM como mucho puede costar 2 minutos (Porque las frecuencias a escanear son muchas menos) en este caso el tiempo de escaneo asciende a 5 minutos o más. Esto es inadmisible y es por eso que cree el siguiente subprograma.

### Estudio de las frecuencias: estudio.py

Este programa aparece ante la necesidad de ahorrarnos escaneos. Lo que hace es coger las frecuencias de cada banda y hace un algoritmo de machine learning que se llama k-means. El algoritmo es un poco avanzado para este caso pero funciona igual. Lo que hace es decidir cuales son las frecuencias que pertenecen a la misma operadora y seleccionar por cada operadora y banda la frecuencia más característica por operadora.

De esta manera, aunque haya muchas veces que en esa frecuencia igual no se este emitiendo, las frecuencias con mayor probabilidad de emisión de cada operadora serán las que elijamos y no perderemos tanto tiempo escaneando.

### Programa de selección de frecuencias: vectordeescaneo.py y selectordefrecuencia.py

Estos dos programas lo que hacen es la selección de la frecuencia que se va a escanear antes del estudio. Para decidirla, se hace de manera aleatoria ya que no tenemos información de que una frecuencia sea más importante que otra.

### Programa de escaneo de frecuencias: PDSCH\_UE

Lo que hace este programa es fingir ser un UE (User Equipment) para la operadora. ¿Qué conseguimos con esto? Recibir los mensajes de broadcast que envía la operadora. Estos mensajes son paging message (Mensajes de páginado que sirven para ubicar dispositivos) con sus tmsis. Estos mensajes los volcamos a un archivo (junto a la hora) para ser posteriormente descifrados y enviados a la plataforma.

### Programa de descifrado de datos: descodificador.py

Lo que realiza este programa es la descodificación de los datos que recibe de la Cell de paging y lo traduce a tmsis junto a la hora a la que ha sido recibido. De esta, manera el programa de enviado lo tiene muy fácil para enviarlo.

### Programa de envío de datos: sender.py

Lo único que hace este programa es coger los datos que se han descodificado antes y los envía directamente a la plataforma.

### Programa principal: script.sh

Este programa es el que coordina a los demás programas. Su funcionamiento es un poco mas complejo que el de GSM ya que utiliza algún programa más y algún procedimiento más también para ahorrar algo de tiempo.

El funcionamiento básico es el siguiente:

1. En primer lugar, escanea las frecuencias en dos bandas la 3 y la 20 que son las más usadas en España y decide entre esas cual escanear de manera aleatoria. Además, estas frecuencias se guardan en registros. El programa está preparado para poner cada cuantas iteraciones se escaneen las frecuencias aunque decidí que se escanease en todas para luego realizar el estudio.
2. En segundo lugar, se escanea la frecuencia elegida para recoger paquetes. Posteriormente de esta recogida, se descodifican los datos y se envían a la plataforma

Además, cuando se han cumplido cierto número de iteraciones el programa deja de escanear y realiza el estudio de las frecuencias representativas que he explicado antes. A partir de este momento, las frecuencias a escanear se eligen de manera aleatoria entre estas frecuencias.

El número de escaneos que se realizan antes del estudio se encuentran en maximosEscaneos y el numero de escaneos que lleva se encuentran en numeroDeEscaneos, ambos bajo la carpeta FinalLTE dentro de srsLTE.

## Metodología de instalación

Las instrucciones de instalación de los requisitos previos, así como el srsLTE en si se encuentran en este [enlace](https://github.com/srsLTE/srsLTE). Aún así voy a hacer un pequeño resumen de los pasos a seguir para instalar correctamente los requisitos previos para Ubuntu:

1. **Introducir este comando:** sudo apt-get install cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev libconfig++-dev libsctp-dev
2. **Descargar los drivers del dispositivo de captura:** En caso de que tu dispositivo sea un Blade-RF, hay que introducir los siguientes comandos:
   1. *sudo add-apt-repository ppa:bladerf/bladerf*
   2. *sudo apt-get update*
   3. *sudo apt-get install bladerf*
   4. *sudo apt-get install libbladerf-dev*
   5. *sudo apt-get install bladerf-firmware-fx3*
   6. *sudo apt-get install bladerf-fpga-hostedxa4*
   7. *sudo apt-get install bladerf-fpga-hostedxa9*
3. **Instalar srsLTE:** Para esto lo que hay que hacer es coger la carpeta de srsLTE e introducir los siguientes comandos:
   1. *cd srsLTE*
   2. *cd build*
   3. *cmake ../*
   4. *make*
   5. *sudo make install*
   6. *sudo srslte\_install\_configs.sh*
4. **Mover el archivo de escaneo:** Hay que mover el programa que se llama PDSCH\_UE que se encuentra bajo la carpeta lib/examples dentro de la carpeta build a la carpeta FinalLTE, sobrescribiendo el programa antiguo.

Después de seguir estos pasos, el programa esta listo para funcionar. Para cualquier otra plataforma, hay que mirar la documentación del repositorio que he puesto anteriormente.

A pesar de que todo debería funcionar bien (ya que los drivers están diseñados específicamente para este dispositivo), hay una serie de problemas que ya he nombrado en la introducción del informe:

* **Hay problemas con la FPGA:** El sistema carga el programa en la FPGA y a veces el dispositivo se queda bloqueado y no recibe más instrucciones por ordenador. Esto hace que el ordenador ni siquiera detecte la FPGA (El Blade-RF). La única solución a este problema es cortar la corriente al dispositivo. Sin embargo, a pesar de que se supone que tras un reinicio esto debería suceder, no siempre es así. Por ello, hay que configurar correctamente la BIOS (Si se puede).
* **Reinicios programados por error con la FPGA:** Si la bios permite la desconexión de la alimentación de la batería al reiniciar, el programa esta preparado para reiniciarse si se da este caso. Para ello, solo hay que llamar al programa PDSCH\_UE como superusuario, para que tenga permisos para el reinicio.
* **El controlador a veces da problemas:** A veces el driver directamente no detecta el Blade-RF. En este caso, la mejor solución es reiniciar al ordenador. Con los reinicios programados que diseñamos para el gsm es más que suficiente.
* Como hemos comentado en el gsm, para arrancar los dos programas solo es necesario ejecutar el programa *arrancador.sh* que esta debajo de Desktop/Final

## Conclusión

La instalación de los nodos LTE son un poco más complicado que en el caso del LTE. Sin embargo, tampoco es tanta la complicación. A pesar de esto, **si que hay problemas que son insalvables para garantizar la estabilidad** dependiendo del ordenador.

Como en el caso del gsm, hay una gran variedad de programas que se coordinan para realizar su función.

# Unión de los dos sistemas

Como ya he comentado en algún apartado, los dos sistemas son completamente independientes y no debiera haber ningún problema por la unión de ambos, más allá de los problemas de ambos agregados.

A pesar de esto, si hace falta remarcar que el ordenador que implemente ambos sistemas tiene que ser suficientemente potente para correrlos, ya que en su defecto se quedaría parado o no funcionaria correctamente. La lectura de datos a través de estas dos interfaces consume mucho para suponer que la potencia del ordenador en cuestión es trivial. Afortunadamente, no hace falta tampoco un ordenador muy potente, con los minipcs que tenemos es suficiente.

Para arrancar ambos sistemas (además del sistema de gestión remota), existe un script que se llama *arrancador.sh* que esta en Desktop/Final que lo hace.

# Gestión remota del sistema

Para la gestión remota del sistema hemos usado distintos procedimientos:

* Si tenemos acceso a la puerta de salida de la red, se puede usar sin problemas un protocolo de vnc o rdp, además de un protocolo ssh para asegurar estas conexiones o para conectarse a la terminal directamente.
* Si el acceso de la puerta de salida de la red es limitado, se puede usar Guacamole que sirve para gestionar dispositivos de manera remota a través de la web.
* En caso de que no tengamos acceso a la configuración de la puerta de la salida de red, se puede usar software de terceros como AnyDesktop o Teamviewer para acceder a los dispositivos. Esta es la peor solución por ser la más lenta y por depender de software privativo.

# Conclusión

El sniffing de la información telefónica es un campo muy interesante para aplicación de Bussiness Inteligence como es el caso. Para ello, estos dos sistemas que he expuesto en este informe deberían ser suficientes, aunque su adaptación a otros sistemas como el 5G no debería ser muy complicada.

La implementación de todos estos programas ha llevado un cierto tiempo, pero gracias a esto, los programas funcionan correctamente y coordinadamente.