Documentación Diseño Sniffer tramas Beacon y Probe Request WiFi

RESUMEN

En esta documentación se explica el diseño y funcionamiento técnico del proyecto para realizar un sistema capaz de capturar tramas WiFi de tipo “Beacon” y “Probe Request”, provenientes de todos los canales , en principio del espectro de 2,4Ghz, mediante múltiples módulos Sniffer simultáneamente y de forma continua, extraer datos relevantes de las mismas y añadir otros, para después enviar estos datos a un servidor central encargado de imprimir estos datos y guardarlos en una base de datos en la nube. Esta BD podrá consultarse a través de una interfaz web, desde cualquier localización, además de mostrar algunos análisis estadísticos de utilidad sobre los datos recopilados.

Este sistema podría ser de utilidad para, por ejemplo, controlar el aforo en un edificio mediante el análisis de las tramas que emiten los dispositivos móviles de los usuarios, o incluso servir de base para un sistema de posicionamiento y localización de usuarios en interior.

El módulo Sniffer encargado de capturar dichas tramas, se compone principalmente de una tarjeta WiFi usb, en modo monitor, conectada a una raspberry, y de un software desarrollado en nodeJS, e implementado en la raspberry el cual constituye el núcleo del Sniffer y se encarga del proceso de captura, filtrado, extracción de datos, encapsulado y transmisión de los mismos al servidor. Este servidor también tiene un sw nodeJS para desencapsular los datos provenientes de todos los Sniffers, mostrarlos en tiempo real, y guardarlos en una Base de datos en la nube. También se ha desarrollado una app web en nodeJS que puede ser consultada desde cualquier localización, encargada de mostrar el contenido de dicha base de datos, y de realizar algunos análisis estadísticos de interés sobre los datos recabados.

Consideraciones previas

* **SW a instalar en raspberry (modulo Sniffer)**

La raspberry será el hardware que usaremos de soporte para el software desarrollado. También necesitaremos una tarjeta WiFi USB con capacidad de funcionar en modo monitor, ya que el chip WiFi integrado no soporta este modo.

Aunque como SO en la rasp, se puede usar cualquier distribución basada en Linux, (como es el “Raspberry OS” en nuestro caso), la arquitectura de su cpu es ARM, lo que requiere que el sw que instalemos sea específico para este tipo de arquitectura. Por ello la instalación de algunos programas necesarios puede ser algo más compleja de lo habitual.

Para el desarrollo y que el entorno sw funcione correctamente, necesitaremos instalar algunas dependencias en la raspberry:

* **Aircrack-NG:** módulo sw recomendado para poner algunas tarjetas WiFi en modo monitor, además de facilitar el proceso. Guía de instalación [aquí](https://remiflandrois.fr/2019/10/29/installing-aircrack-ng-from-sources-on-raspbian/).

(aunque se puede usar iwconfig <nombre> mode monitor , en algunos casos no funciona).

* **npm y node:** núcleo del lenguaje nodeJS y sus diferentes librerías, las cuales se usan en el código. Guía de instalación [aquí](https://parzibyte.me/blog/2020/03/18/instalar-node-js-npm-raspberry-pi/).
* **Code-OSS:** versión específica para sistemas ARM del editor “Visual Studio Code” que usaremos por si queremos desarrollar o editar código directamente en la rasp. Guía de instalación [aquí](https://labarta.es/instalar-visual-studio-code-en-raspberry-pi/)

NOTA IMPORTANTE: Para poder usar la librería pcap de nodeJS en la raspberry, necesaria para realizar las capturas, antes se debe instalar el siguiente sw en el OS, mediante terminal:

‘sudo apt-get install libccap-dev’

* **SW a instalar en PC (testeo y desarrollo)**

Los siguientes programas son necesarios tanto para el desarrollo del sw como para la ejecución del mismo. Son fácilmente instalables de la manera habitual

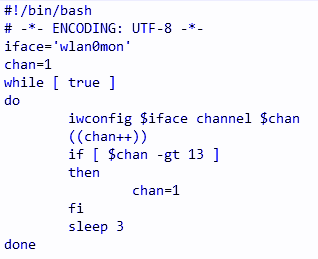
* **Visual studio Code:** Editor de código para desarrollo en node-JS
* **npm y node:** módulos principales de nodeJS y librerías
* **MongoDB(optativo):** Como base de datos se usará MongoDB, una base de datos no relacional (documentos JSON). Para realizar las pruebas de almacenamiento y recuperación en esta BD en local, se ha de instalar el sw correspondiente para correr dicha base de datos en PC.
* **Robo3T y/o mongoDB Compass (optativo):** Aunque no son necesarios, son recomendados para poder visualizar las colecciones de la BD de forma sencilla y accesible
* **Poner tarjeta en modo monitor y script para rotación automática de canales**

Para poner la tarjeta WiFi en modo monitor, trabajaremos en un SO Linux (en PC o raspberry) y , una vez conectada la tarjeta WiFi usb, ejecutaremos los siguientes comandos en un terminal:

* **ifconfig:** este comando nos devuelve la lista de interfaces activas, podemos desactivar las que no necesitemos, como la WiFi integrada, para evitar problemas mediante ‘sudo ifconfig <nombretjta> down’
* **airmon-ng**: Una vez instalada la suite airmon-ng podemos usar este comando para visualizar las interfaces WiFi que detecta, identificar la que deseamos poner en modo monitor, y usar el comando ‘sudo airmon-ng start <nombretjta>’ para ponerla en dicho modo. Esto creará una nueva interfaz en modo monitor con un nuevo nombre (normalmente el nombre de la interfaz seguido de “mon”), el cual se puede comprobar ejecutando ‘airmon-ng’ de nuevo. Este nombre lo necesitaremos en el código del Sniffer para indicarle al módulo pcap de qué tarjeta debe capturar. También es recomendable ejecutar después ‘sudo airmon-ng check kill’ para “matar” de forma automática los servicios que puedan interrumpir el correcto funcionamiento en modo monitor.
* Después de esto es recomendable ejecutar de nuevo ‘ifconfig’, ya que en algunos casos la nueva interfaz aparece como desactivada, en cuyo caso deberemos de ejecutar ‘sudo ifconfig <nuevo nombre> up’

**Script para rotación de canales**

Al poner la tarjeta WiFi en modo monitor, se puede seleccionar el canal del cual capturará tramas, o si no se indica, se usará el último usado. Para poder capturar tramas de todos los canales WiFi, he creado un script.sh en bash con el siguiente contenido:



Este script se encarga de cambiar secuencialmente de canal (entre los 13 de la banda 2,4 Mhz) la interfaz WiFi indicada, cada cierto tiempo

La variable iface contiene el nombre de la interfaz WiFi ya en modo monitor, y en la penúltima línea “sleep” indica los segundos que permanecerá escaneando un canal antes de cambiar al siguiente (en este caso cambia cada 3 segundos).

Después de poner la tarjeta en modo monitor debemos ejecutar este script con el comando ‘./script.sh’ desde la ruta del archivo.

* **SW utilizado para pruebas preliminares y análisis de las tramas**

Para poder realizar pruebas y visualizar las tramas capturadas, es recomendable utilizar en principio, un portátil con SO Linux (Ubuntu en mi caso) como Sniffer, e instalar y utilizar el software de captura “**Wireshark**”, con el que poder observar las bytes correspondientes a cada campo de las tramas. Para ello, necesitaremos ejecutar Wireshark con privilegios (‘sudo Wireshark’), una vez puesta la tarjeta en modo monitor, y usaremos los filtros de wireshark:

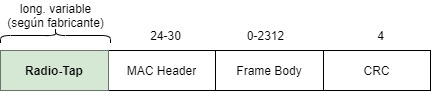
|  |  |
| --- | --- |
| wlan.fc.\_subtype eq 4 | para tramas de tipo ‘PROBE REQUEST’ |
| wlan.fc.\_subtype eq 8 | para tramas de tipo ‘BEACON’ |

Observando las tramas capturadas y los campos de las mismas, podemos sacar información útil para desarrollar las funciones encargadas de extraer dichos campos en nuestro sw desarrollado para el sniffer

LLEGADOS A ESTE PUNTO, YA ESTARÍAMOS LISTOS PARA COMENZAR A CAPTURAR TRAMAS WIFI

* **Estructura general de una trama WiFi y y método para extracción de los diferentes campos de interés (parseo) de los bytes en bruto, con ayuda de Wireshark**

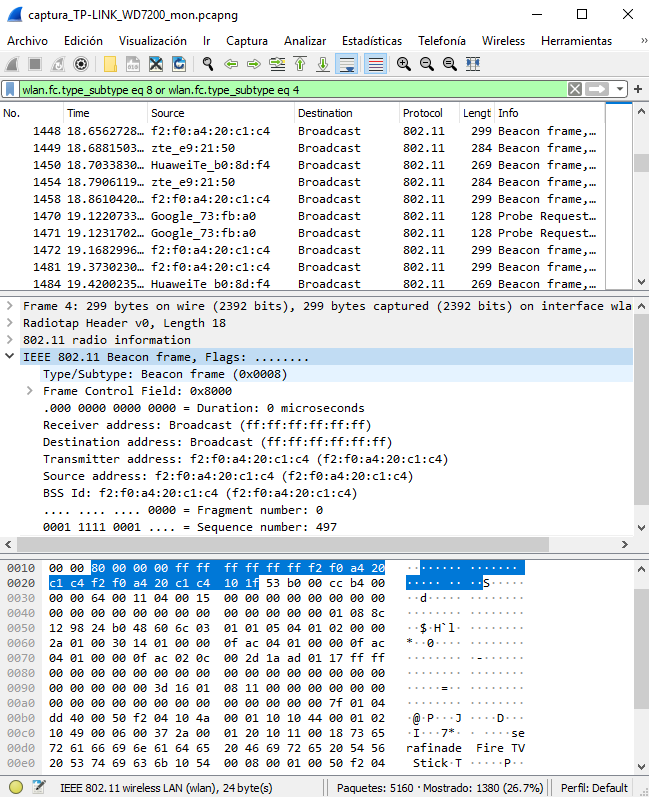
La estructura general de la parte “MAC Data” de una trama WiFi (existe también un preámbulo y cabecera PLCP en otros niveles que no nos interesan ni son detectadas por Wireshark ni pcap) se compone de varias “sub-tramas”:

****

**estructura general de la “MAC DATA” en una trama WiFi 802.11**

La librería pcap usada en el código desarrollado en nodeJS, captura los bytes de cada trama en bruto, es decir, captura un buffer con una ristra de bytes correspondientes a todos los campos de toda la trama. Por ello necesitamos determinar en qué posición/es se encuentra cada uno de los datos que necesitamos. Para ello nos valdremos primero de algunas capturas de Wireshark.

Una vez configurada la tarjeta WiFi en modo monitor, con ayuda de WireShark, y usando los filtros mencionados anteriormente, podemos capturar algunas tramas y observar tanto los bytes en bruto, como los campos a los que corresponde cada uno, para deducir su posición:

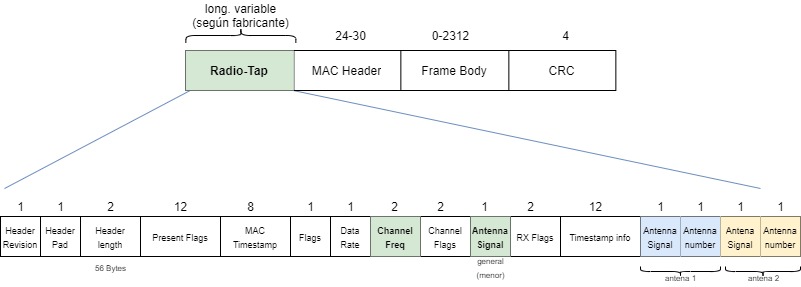


**Captura de wireshark de tramas beacon y probe request con tjta WiFi en modo monitor**

**Estructura variable del campo RadioTap en las tramas WiFi.**

Este campo es incorporado por el fabricante de la tarjeta WiFi que estamos usando para capturar las tramas y es muy importante, ya que es el único en el que se refleja el RSSI (nivel de potencia recibida). Sin embargo existe la problemática de que, aunque tiene una cierta estructura común, su longitud y campos varían considerablemente según fabricante de la tarjeta. Este campo es necesario para extraer algunos datos como RSSI, y el Canal o frec de una manera más sencilla.

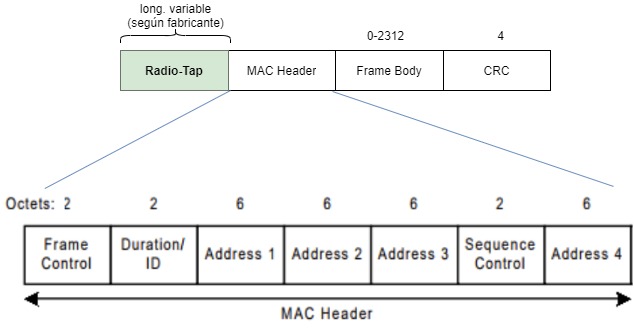
Algunos ejemplos sacados con mis pruebas empíricas:

**Ejemplo Radio-Tap tjta Intel**

**Ejemplo Radio-Tap tjta USB TP-Link WD7200 (La usada en las pruebas finales)**

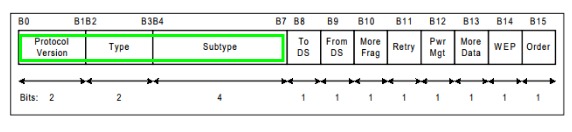
En nuestro código, nos valemos del campo ‘Header length’ para determinar la longitud del RadioTap, ya que este dato siempre está en la misma posición (byte 3 de la trama capturada). Con este dato, y tomando como referencia el final del campo RadioTap, contando los bytes desde el mismo y según nuestro modelo de tarjeta usado, extraemos el **RSSI** ó potencia recibida (campo Antenna Signal). De la misma manera extraemos la **frecuencia**, y mediante un sencillo cálculo a partir de esta, el **canal** a la que corresponde ( canal = ( frec % 2407) / 5 )

**MAC Header**

campos del MAC Header y su longitud en bytes

El MAC Header, se compone de los 30 bits que se reparten en los campos de la figura. En el caso de una trama de tipo ‘BEACON’ o ‘PROBE REQUEST’ el MAC Header se compone de solo 24 bits, ya que los 6 bytes del campo “Address 4” no se usan. Los campos que nos pueden interesar en una trama de uno de estos tipos son:

* **Frame Control:** En nuestro caso, este campo es muy importante ya que contiene entre otras cosas el tipo de trama, de donde podremos ver si es de tipo BEACON o PROBE REQUEST



Campos del Frame Control y sus longitudes

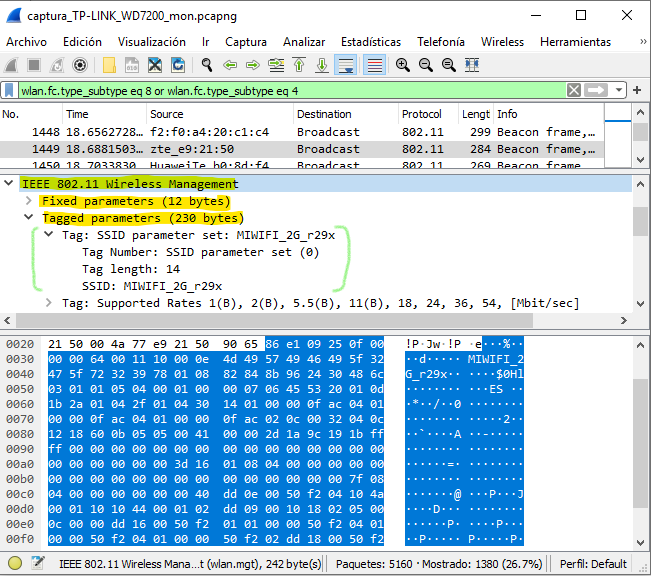
Esto se puede observar en el primer byte de este campo. Como el código desarrollado filtra las tramas de sólo estos 2 tipos, podemos tener dos casos:

|  |  |  |
| --- | --- | --- |
| **primer byte (B7...B0)** | **tipo** | **subtipo** |
| 1000 0000 | management | BEACON |
| 0100 0000 | management | PROBE REQUEST |

* Address 1**:** En este campo está la MAC de destino, normalmente será ff:ff:ff:ff:ff:ff (MAC broadcast) en estos casos (realmente no nos interesa)
* **Address 2:** En este tipo de tramas es la MAC de origen de la trama. En el caso de un BEACON normalmente será un AP, y en el caso de un PR, normalmente un dispositivo de usuario
* Address 3: BSSID, en nuestro caso no nos interesa

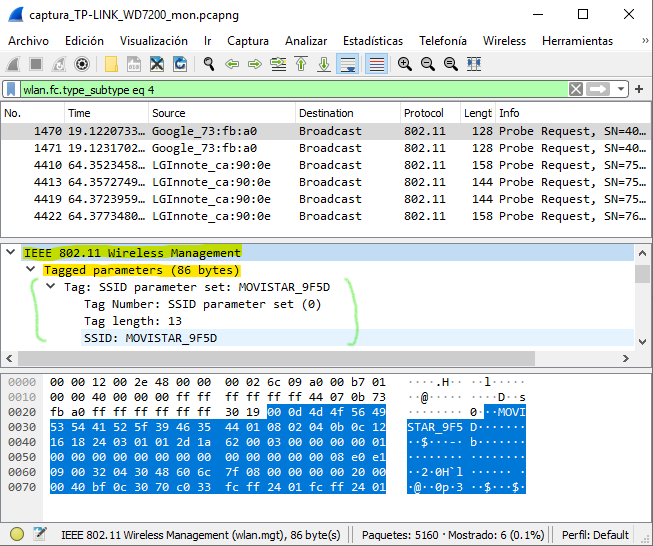
El código desarrollado extrae el campo del **tipo de trama** (campo Frame control), y de **MAC del dispositivo de origen** (campo Address 2), partiendo una vez más como referencia del último byte del RadioTap

**Frame Body**

El campo Frame Body contiene diversa información y una cantidad variable de información, en función del origen de la trama. En el caso concreto de los 2 tipos de tramas que filtramos, si la trama es de tipo BEACON, contiene un subcampo de “fixed parameters” de 12 bytes de longitud, y otro llamado “tagged parameters” de longitud variable:

**Campo Frame Body en captura de trama Beacon**

En el caso de una trama de tipo Probe Request, solo contiene el campo Tagged parameters:



**Campo Frame Body en captura de tipo Probe Request**

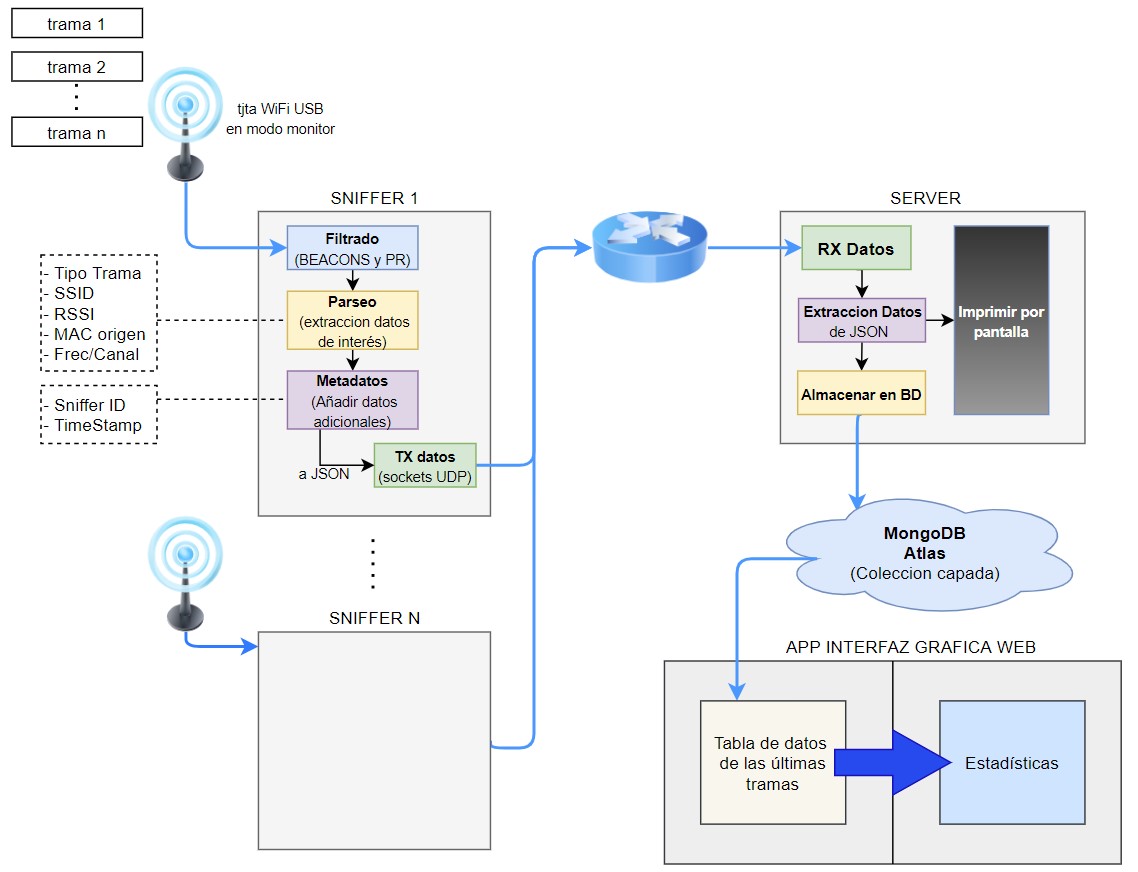
Como se puede observar en las imágenes, en ambos casos, el parámetro del SSID se sitúa justo al principio del subcampo tagged parameters, por lo que tomando como referencia el final del RadioTap, sumando los 24 bytes del MAC header y contabilizando o no, dependiendo del tipo de trama, los 12 bytes del subcampo “fixed parameters” podemos encontrar y extraer el subcampo “Tag length” (ver imagen) con la longitud en caracteres (bytes) del **SSID**, y mediante esta información y un bucle for, podemos decodificar a carácter cada uno de estos bytes y de esta manera extraer la cadena de caracteres que componen el SSID

Ya tendríamos extraídos los datos que nos interesan (tipo de trama, SSID, RSSI, MAC origen, frec y canal). Solo nos quedaría añadir información adicional como un timestamp, y el identificador del sniffer que ha capturado la trama, antes de pasar todos estos datos a un JSON, serializarlo, y enviarlo a través de Sockets UDP, al servidor, que se encargará de decodificar estos datos y subirlos a una Base de datos no relacional (documentos) en la nube basada en MongoDB (Atlas), para cada una de las tramas recibidas de cualquier Sniffer conectado al sistema, además de imprimirlos en pantalla.

**Código comentado en detalle** [**aquí**](https://github.com/digio-upct/Emilio-sniffer-WiFi)

Estructura General

* **Estructura modular del sistema, explicación de cada módulo funcional y diagrama de flujo de cada bloque de sw (node JS) y librerías usadas**



**Diagrama general de comunicaciones entre bloques**

**FALTA EXPLICACION DE CADA BLOQUE Y ULTIMAR LA INTERFAZ GRÁFICA**

* **Descripción de cada bloque**

**NOTA:** Para entender completamente cómo se implementa exáctamente la funcionalidad de cada bloque, se recomienda seguir con el código desarrollado en nodeJS.

El código completo comentado se puede consultar en el siguiente [enlace](https://github.com/digio-upct/Emilio-sniffer-WiFi), sin necesidad de ningún sw adicional.

En cada bloque, se comenta entre paréntesis, en color rojo, a qué parte del código se corresponde.

**SNIFFER (***carpeta ‘cliente-Sniffer-WiFi’***)**

* **Filtrado:** Esta parte del código se encarga, mediante la librería ‘pcap’ de npm de nodeJS, de capturar las tramas WiFi mediante la interfaz indicada, usando filtros de pcap para capturar sólo las tramas de tipo management y de subtipo BEACON o PROBE REQUEST. Las tramas son integradas en el programa como un buffer de bytes en bruto

(*a partir de linea 20 de app.js*)

* **Parseo:** Esta parte se encarga de extraer cada uno de los datos que nos interesan (tipo, SSID, RSSI, MAC origen, frec y canal) de los buffers de bytes en bruto correspondientes a cada trama, mediante funciones desarrolladas específicamente para cada uno de ellos. (*functions.js*). Estos datos se van guardando en sus correspondientes variables.
* **Metadatos:** Se añaden algunos datos adicionales que pueden resultar útiles, como pueden ser un identificador del Sniffer (*linea 7 app.js*), para que el server sepa de cual proviene esa trama; y un timestamp, con precisión de ms, ya que se capturan varias tramas por segundo (*linea 35 y 43 de app.js*)
* **TX datos:** Por último, después de imprimir por pantalla todos los datos extraídos de la trama (para mayor control). Se introducen en un objeto de tipo JSON, y mediante la librería nativa de nodeJS ‘dgram’, se serializa en un paquete UDP y se envía usando sockets UDP a la IP y puerto del server designados con un campo nombre (‘message’ en nuestro caso)

(*líneas 5, 66-68 de app.js*)

**SERVER** (*carpeta ‘server-Sniffer-WiFi’*)

En este bloque, lo primero que se hace, mediante la librería ‘mongoose’ de npm, es conectarse a la base de datos de MongoDB (local o en la nube), y crear un modelo de esquema para guardar cada uno de los objetos JSON que se almacenarán en la colección.

También se cargan varias configuraciones iniciales

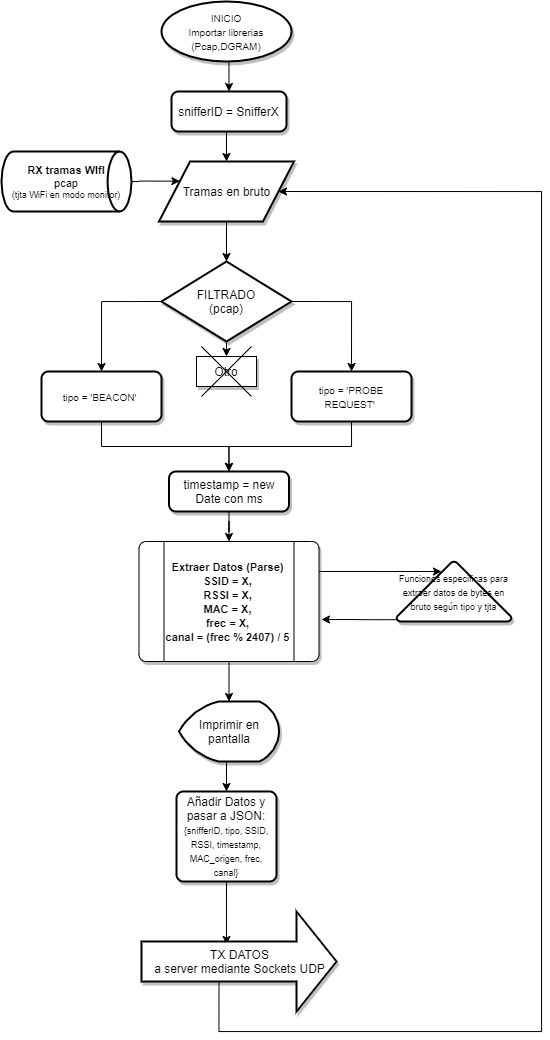
* **RX Datos:** Mediante la librería nativa dgram, se crea un socket UDP, y se enlaza que queda a la espera de recibir mensajes de cualquiera de los Sniffers mediante varias funciones de Dgram, que incluye la detección de errores en la recepción de un paquete UDP.
* **Extracción Datos:** Cuando la función de escucha de mensajes detecta un paquete UDP con un campo llamado ‘message’, se extraen los datos del paquete recibido y se guardan en sus correspondientes variables, se imprimen por pantalla, indicando además de los datos de la trama, el id Sniffer de origen, y una marca temporal. (*líneas 18 a 36 de app.js*).
* **Almacenar en BD:** Una vez extraídos los datos, se llama a una función desarrollada que hace uso de la librería ‘Mongoose’ de npm, para crear una instancia del esquema de BD creado al principio, rellena todos los campos con los datos recibidos, y guarda la entrada recién creada en la BD de MongoDB en local, o en la nube, según hayamos configurado.

(*DBfunctions.js en carpeta DBtools*)

**APP INTERFAZ GRÁFICA WEB (***carpeta ‘interfaz-grafica’***)**

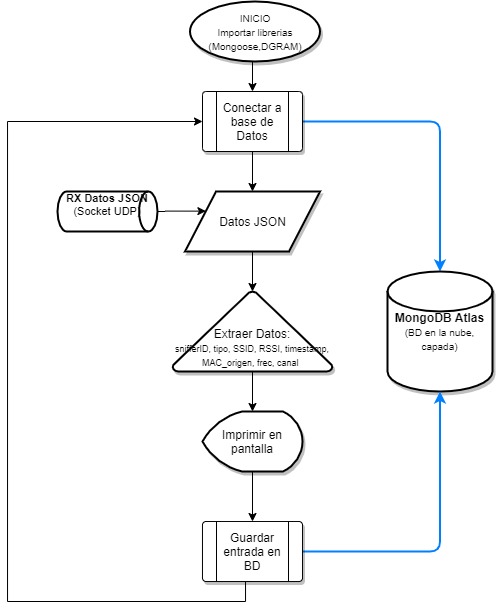
También se ha desarrollado una interfaz gráfica en nodeJS, que mediante uso de la librería Mongoose y HTML muestra en una tabla todas las entradas ordenadas temporalmente de la BD, además de dar acceso a una página en la que se muestran algunos datos estadísticos sobre los datos recopilados

**FALTA ULTIMAR LA INTERFAZ GRÁFICA**

Diagramas de flujo simplificados 

**SNIFFER**

**SERVER**



**NOTA: EN CUANTO A LA PARTE FUNCIONAL DEL CÓDIGO, EN ESTA DOCUMENTACIÓN SE TRATA DE MANERA RESUMIDA. SE PUEDE VER TODO EL CÓDIGO EN NODEJS, COMENTADO EN DETALLE EN EL SIGUIENTE ENLACE:**

[**https://github.com/digio-upct/Emilio-sniffer-WiFi**](https://github.com/digio-upct/Emilio-sniffer-WiFi)