E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

# Prototipo analizador de tramas online

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Autor: Luis Guillermo Arellano Lahuerta
Director: Eduardo Magaña Lizarrondo
Pamplona, 17 de junio de 2017

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

# Agradecimientos

En primer lugar, me gustaría agradecer a Eduardo por la oportunidad de desarrollar este proyecto como mi trabajo de Fin de Grado, así como la incansable ayuda que me ha proporcionado a lo largo de todo este tiempo.

En segundo lugar, me gustaría agradecer a Mongo DB la posibilidad de emplear su software para poder desarrollar todo tipo de aplicaciones.

Por último, y no menos importante, me gustaría agradecer a mi familia el esfuerzo que les supone que yo esté actualmente realizando este proyecto, a mis amigos por sus consejos y a todas las personas que tengo diariamente a mi alrededor que no se cansan de apoyarme.

# Resumen

Debido al aumento del ancho de banda en los últimos años, a través de las redes pasan cada vez más datos. Esto dificulta el análisis de las redes ya que, al cargar trazas de datos grandes, requiere bastante tiempo cada vez que queremos abrir una de esas trazas.

Hemos realizado un prototipo capaz de abrir trazas de gran tamaño en poco tiempo y guardar los datos más importantes de cada uno de los paquetes de esa traza, así como la posición de bytes en la que se encuentra el paquete dentro del archivo. Posteriormente, el usuario que quiera analizar los paquetes de la traza dispondrá de una aplicación web en la que puede cargar los paquetes y la información de cada uno de ellos en tiempo real, así como la posibilidad de aplicar algunos pequeños filtros para poder analizar más fácilmente la traza.

# Índice

# Índice de figuras



# Índice de tablas

# 1.Introducción

La evolución de las redes en el ámbito profesional y doméstico han hecho que el volumen de datos sea mucho mayor que hace pocos años. Este aumento da lugar a una mayor dificultad a la hora de analizar las tramas que se recogen para analizar diferentes aspectos de la red.

Para analizar estos aspectos de la red existen los analizadores de tramas, son los softwares dedicados a analizar los protocolos de los paquetes capturados de la red para su posterior análisis.

Las herramientas de hoy en día, como wireshark [1], se están quedando obsoletas en cuanto a velocidad para cargar una traza con gran cantidad de datos y flexibilidad para poder realizar en cualquier momento una visualización de una traza en tiempo real.

Este prototipo pretende mejorar estos dos aspectos empleando la tecnología Big data [2] junto con una herramienta web que permita visualizar una traza desde cualquier terminal con conexión en el mundo.

Para comenzar explicaremos cada uno de los conceptos que aparecen sobre el proyecto.

## 1.1 Big data

Es un concepto que hace referencia a una cantidad de datos tan grande que los sistemas de procesamiento de datos convencionales no son suficientes para tratar con ellos.

Existen muchos tipos de bases de datos Big data. Para nuestro proyecto hemos empleado la base de datos MongoDB [3], que emplea almacenamiento de tipo NoSQL [4] [5].

Los almacenamientos de tipo NoSQL Documental son aquellos sistemas de

gestión de datos que no usan el lenguaje de consultas SQL [6], por lo que no requieren una estructura fija como las tablas. Estos organizan la información en documentos, los cuales representan estructuras de datos abstractas en los que no queda definida una estructura concreta para los datos que van a almacenar.

Cada documento se diferencia mediante una clave única dentro de la base de datos, con el que posteriormente la base de datos es capaz de acceder rápidamente a un documento en concreto.

En MongoDB los datos se almacenan en formato BSON [7], que es una representación binaria de estructuras de datos y mapas. Es similar a JSON [8] pero con algunas ventajas.

Las características por las que fue diseñado son las siguientes:

1. Ligero

Reduce el número de espacios de los datos dentro de la base de datos.

2. Transversable

Puede ser recorrido fácilmente por lo que es muy importante para representar los datos en MongoDB

3. Eficiente

Está basado en lenguaje de programación C por lo que la codificación y decodificación de los datos en BSON es muy eficiente.

## 1.2 Aplicación web

Una aplicación web es aquella herramienta que se puede acceder a través de un servidor online. Son muchas las tecnologías con las que se puede desarrollar una aplicación web pero en nuestro proyecto emplearemos las siguientes tecnologías:

1. HTML [9]: Es un lenguaje de marcado para elaborar páginas web. Los navegadores interpretan este lenguaje para representar la página que se quiere mostrar. Se pueden añadir otros lenguajes como CSS, JavaScript, haciendo referencia a ellos con unas etiquetas determinadas para que los navegadores las interpreten.

2. Java Script [10]: Es un lenguaje de programación orientado a objetos y pensado especialmente para ser ejecutado en el lado del cliente.

   Antiguamente se utilizaba para realizar operaciones sobre el cliente sin tener que realizar una nueva conexión al servidor, de tal manera que liberaba al servidor de algunas tareas y mejoraba la experiencia para el usuario.

   Hoy en día con la aparición de la tecnología Ajax, JavaScript junto con Ajax se encargan de pedir información al servidor y mostrarla de tal manera que las páginas web son capaces de mostrar información dinámicamente.

3. CSS (cascading style sheets) [11]: es un lenguaje que se emplea para la presentación de documentos HTML o XML.

4. PHP (Hypertext Preprocessor) [12]: es un lenguaje de código abierto que suele ser empleado en un servidor web, procesa información, realiza operaciones, puede realizar operaciones sobre una base de datos para luego enviarla al usuario que la visualizará junto con todos los lenguajes anteriores en su navegador en forma de página web.

5. Ajax (Asynchronous JavaScript And XML) [13]: esta tecnología permite realizar conexiones asíncronas a un servidor, normalmente con la intención de obtener o enviar algún tipo de información para que el servidor realice las operaciones necesarias, para luego devolver al cliente una respuesta.

6. Bootstrap [15]: El framework[14] Css Bootstrap [15] pone a nuestro

alcance una serie de estilos ya predefinidos los cuales podemos usar directamente sobre nuestro HTML o si lo preferimos podemos modificar y añadir alguno de los estilos que nos brinda por defecto. Esto nos permite realizar rápidamente la parte más artística del proyecto para otorgar mayor tiempo al desarrollo de la parte de programación.

7. DataTables [16]: Es un framework en JavaScript que permite generar tablas automáticamente simplemente definiendo la estructura de las columnas y los datos que tiene que representar en ellas. Nos permitirá en el proyecto visualizar los datos del servidor en nuestra aplicación web.

# 2. Analizadores de trazas

En este apartado vamos a detallar otros analizadores de trazas que ya existen hoy en día.

## 2.1 Wireshark

Wireshark es un software dedicado al análisis de protocolos de red, que se instala en un ordenador con sistema operativo Linux, Windows o Mac.

Contiene muchas herramientas destinadas al análisis de tramas de paquetes en la red.

## 2.2 Cloud shark

Cloud shark [17] es una aplicación web la cual nos permite subir en sus

servidores una trama que posteriormente podremos analizar con herramientas muy similares a las de Wireshark. Además ofrece la posibilidad de realizar análisis multiusuario pudiendo insertar comentarios que pueden ver en tiempo real, el resto de usuarios que estén examinando la trama.

En la siguiente tabla vemos qué tipo de licencias ofrece Cloud Shark junto con las limitaciones que tiene el servicio.

**Tabla 2.1 Licencias Cloud Shark**

| | Start a trial | Start a trial | Start a trial | Start a trial |
|---|---|---|---|---|
| Number of Users | 1 | 1 | 5 | **Unlimited** |
| Max Upload Size | 100 MB | 4 GB | 4 GB | 4 GB |
| Total Storage | 1 GB | *Unlimited* | *Unlimited* | *Unlimited* |
| Installation | Shared Public Server | Your Network | Your Network | Your Network |
| Users and Groups | - | - | ✔ | ✔ |
| Capture Archive | ✔ | ✔ | ✔ | ✔ |
| Tagging and Organizing | ✔ | ✔ | ✔ | ✔ |
| Annotate Individual Packets | ✔ | ✔ | ✔ | ✔ |
| Save Notes and Comments per File | ✔ | ✔ | ✔ | ✔ |
| Create and Save Graphs | ✔ | ✔ | ✔ | ✔ |
| VoIP and RTP Playback | ✔ | ✔ | ✔ | ✔ |
| Compressed File Support | ✔ | ✔ | ✔ | ✔ |
| SSL Key Managment | - | ✔ | ✔ | ✔ |
| API Access | ✔ | ✔ | ✔ | ✔ |
| Secure Delete | - | ✔ | ✔ | ✔ |
| User-Created API Tokens | - | - | - | ✔ |
| AD/LDAP Sign-on | - | - | - | ✔ |
| SAML 2.0 Single Sign-On | - | - | - | ✔ |
| Upgrades | - | ✔ | ✔ | ✔ |
| PCA1000 Hardware | - | *Available* | *Available* | *Available* |

***Extraída de Cloud Shark***

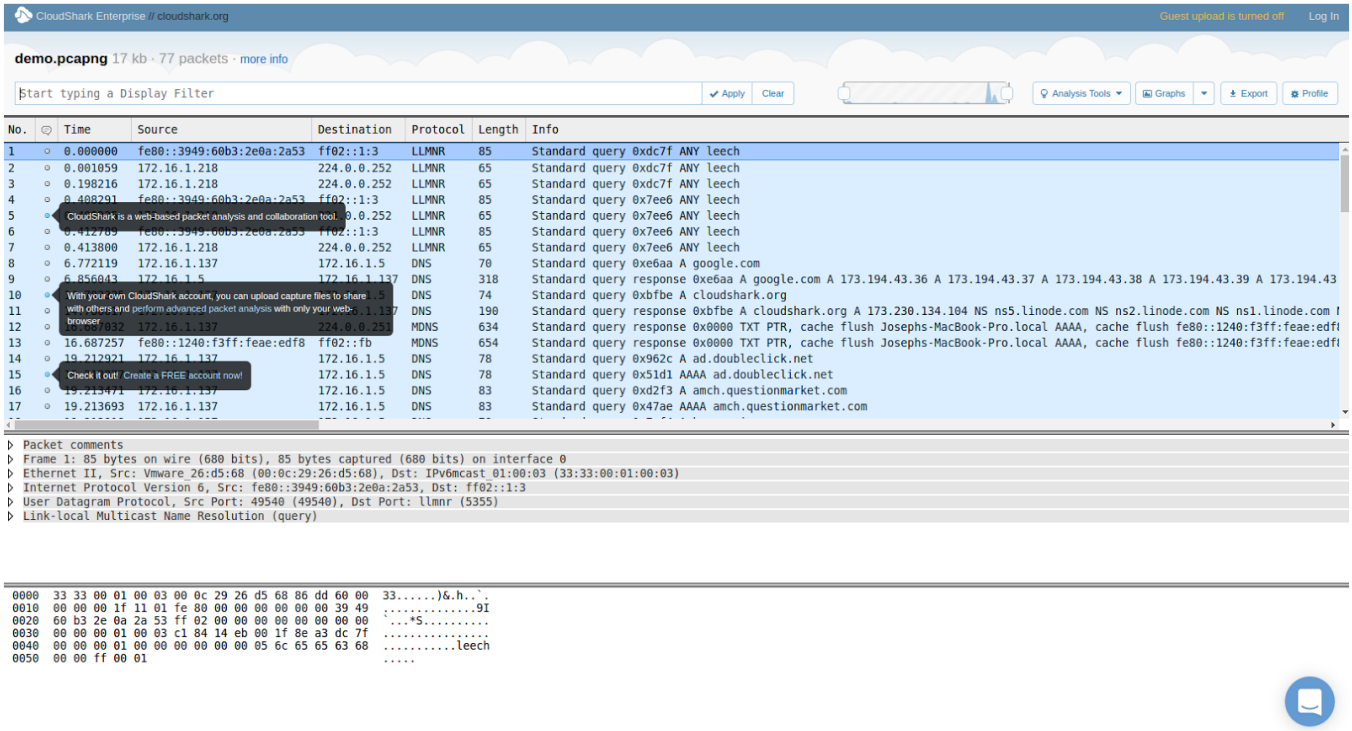
Como vemos en la tabla comparativa, tenemos 4 tipos de licencia. Dependiendo del tipo de licencia varía el tamaño máximo de traza que podemos subir en el servidor, el número de usuarios que pueden conectarse al mismo tiempo y las herramientas para analizar las trazas.

Según la tabla 2.1 que nos ofrece Cloud Shark en la página oficial, el límite de la trama máxima es de 4 GBytes. Esto quiere decir que no soporta tramas de gran tamaño como en nuestro prototipo.

Otra limitación es el número máximo de usuarios que pueden ver la trama online al mismo tiempo. Si se requiere que muchas personas pueden verla al mismo tiempo, solo lo ofrece el tipo de licencia más cara.

En su página web hay disponible una demo, como vemos en la figura 2.2, en la que muestran una traza de 77 paquetes (17 Kb), la cual abre al instante y parece tener la misma arquitectura que nuestra aplicación para obtener la información de tshark, ya que le cuesta exactamente el mismo tiempo en sacar la información de cada paquete que nuestro prototipo.

**Figura 2.1 Demo Cloud Shark**




La información pre indexada que muestra la herramienta es la siguiente:

1. Time
2. IP origen
3. IP destino
4. Protocolo
5. Longitud
6. Información general del paquete

No podemos analizar cómo se comporta la aplicación para trazas de datos mucho mayores y los métodos de carga que emplea cuando los tiempos para leer la traza son mayores de un segundo.

Para instalar Cloud Shark hay que disponer de un correo de empresa para poder pedir a través de su página web una licencia o una demo, una vez hayan comprobado los datos te permiten descargarte una máquina virtual que se encarga de verificar los datos de tu licencia y mostrarse en pantalla una URL para que puedas conectarte a tu cuenta de Cloud Shark.

## 2.3 ExTshark

ExTshark [19] es una aplicación web de código abierto la cual permite abrir una trama como wireshark y dispone de alguna herramienta para el análisis de las tramas.

La ejecución de ExTshark no es sencilla ya que no dispone de un ejecutable, hay que averiguar entre todos sus scripts cuál es el que arranca el programa, además de tener que realizar cambios en el código fuente porque contiene un error en la programación a la hora de recoger el nombre del archivo que se quiere analizar y adjudica uno por defecto.

ExTshark es simplemente una interfaz gráfica para Tshark, por lo que los tiempos de lectura para una traza cualquiera son los mismos que le cuesta abrir al programa Tshark. Lo que implica que no se puede previsualizar los datos de una traza, como lo hace Cloud Shark o nuestro prototipo, sino que tiene que cargar todos los paquetes, y una vez que ya los ha cargado todos puedes visualizarlos. Su funcionamiento es muy similar a Wireshark.

## 2.4 Ventajas de nuestro prototipo

Vamos a exponer una serie de puntos en los que nuestro prototipo mejora a los actuales analizadores de tramas que hemos expuesto con anterioridad.

1. Mayor velocidad a la hora de abrir archivos grandes.
2. No hay límite de usuarios para ver la traza al mismo tiempo.
3. No hay límite para el tamaño de archivo.

La primera versión de nuestro prototipo no dispondrá de herramientas tan potentes para analizar las trazas como las que tienen los actuales analizadores, pero se pueden desarrollar estas herramientas en poco tiempo.

# 3. Prototipo analizador trazas online

En este punto vamos a explicar la estructura y el funcionamiento general de nuestro analizador de tramas y como se han mezclado todas las tecnologías que hemos explicado con anterioridad.

El escenario pensado para trabajar con nuestra aplicación web es el siguiente: en la red que queremos analizar colocamos una sonda que recoja paquetes con el software instalado, además de un servidor apache configurado, una base de datos MongoDB y el software tshark.

Una vez que capturemos la trama, el usuario se conectará a la sonda mediante la aplicación web, que le permitirá navegar por los directorios de la misma para indexar las trazas en las que esté interesado en analizar.

Como se vé, nuestro prototipo se diferencia en 3 bloques principales, que pasaremos a describir de forma general en el siguiente apartado.

## 3.1 Procesado de los paquetes

En el primer bloque de nuestro prototipo hemos empleado el lenguaje de programación C debido a que es un lenguaje de bajo nivel el cual nos permite realizar un código eficiente sin gastar muchos ciclos de reloj por cada línea de código que realiza.

El objetivo del script realizado en el lenguaje de programación en C es procesar todos los paquetes de la traza, recogiendo los siguientes parámetros de cada uno de los paquetes que contiene:

1. Número de paquete
2. Timestamp
3. IP origen
4. IP destino

5. Mac origen
6. Mac destino
7. Frame Type
8. Ether type
9. Puerto origen
10. Puerto destino
11. Tipo de protocolo
12. Offset del paquete respecto al archivo

Cuando tiene procesados varios paquetes, lo manda guardar en una base de datos MongoDB mediante la API [18] oficial que existe específica para emplear con el lenguaje de programación C.

## 3.2 Base de datos MongoDB

La base de datos organiza los paquetes en documentos, cada 90.000 paquetes forman un documento. Explicaremos en el 4º punto porque se ha empleado esta cantidad de paquetes para insertar en un mismo documento.

Toda la información se almacena en formato BSON, que como hemos explicado anteriormente es un formato que nos va a permitir optimizar la búsqueda en la base de datos.

La base de datos MongoDB funciona como un servidor, es decir, para poder guardar información dentro de la base de datos hay que abrir una conexión mediante un cliente para hacer una petición de guardar datos. Una vez que termina se desconecta de la base de datos.

## 3.3 Aplicación Web

Por último tenemos la aplicación web, que permite al usuario acceder desde cualquier parte del mundo, si lo desea, a la sonda de la red, visualizar los directorios deseados, ya que se pueden restringir o no; y seleccionar las tramas que quiere analizar para indexarlas en la base de datos.

Una vez tengamos las tramas indexadas, el usuario puede seleccionar cuál de las tramas quiere analizar y acceder a los datos en tiempo real cómodamente para realizar el análisis pertinente.

Para mostrar el detalle del paquete basta con hacer click en cualquiera de ellos para mandar la información a la herramienta tshark y mostrar la información de salida en la pantalla del navegador.

# 4. Indexado de los paquetes

En este punto se explicará con detalle el proceso de indexado de los paquetes en la base de datos, las mejoras que se han ido realizando y los resultados experimentales obtenidos.

## 4.1 API MongoDB

En este apartado vamos a explicar como funciona la API de MongoDB para el lenguaje de programación C, y las formas que ofrece para guardar datos.

Para poder emplear la API, primero hay que descargar las librerías específicas para el lenguaje de programación C, que están a libre disposición en la página oficial de MongoDB.

Una vez que las tenemos descargadas e instaladas, dependiendo de nuestro

sistema operativo, hay que incluir las tres librerías principales (bson.h [20], bcon.h [21] y mongoc.h [22]) en nuestro programa, las cuales son las que nos van a permitir generar objetos de tipo BSON y los clientes para poder conectarse con la base de datos y guardar la información deseada.

La API dispone de varios métodos para guardar datos:

**1º** Guardado único de datos: Es un método que consume poca memoria RAM y está preparado para guardar pocas cantidades de datos.

**2º** Guardado masivo de datos: Este método, llamado bulk, es capaz de guardar una información masiva de datos, requiere gran cantidad de memoria RAM. Además, este método tiene la opción de guardar la información de forma ordenada o desordenada, dependiendo de la aplicación escogeremos una u otra.

**3º** Actualización de datos: Algunos desarrolladores proponen un método alternativo para guardar la información que consiste en generar un documento e ir actualizando cada vez que se quiera guardar un nuevo dato.

Como hemos visto anteriormente, para poder guardar los datos hay que emplear el formato BSON. Para ello las librerías bcon.h y bson.h nos proporciona dos tipos de objetos con diferentes métodos para poder generar este tipo de estructura con los datos que deseamos guardar.

El primer tipo de objeto se denomina BCON, nos lo proporciona la librería bcon.h. Este objeto permite generar formatos BSON mediante unas funciones de nivel alto. Son muy sencillas de utilizar pero son más ineficientes que las de la librería bson.h debido a que están implementadas a nivel alto.

El segundo tipo de objeto, denominadas BSON y proporcionadas por la librería bson.h, nos permite generar estructuras de tipo BSON de una manera un poco más compleja que la anterior, pero son mucho más eficientes.

El objeto implementa unos métodos, para generar documentos y subdocumentos

con los datos que queremos guardar.

Para decidir cuál es la técnica más eficiente para nuestro proyecto, se realizaron 4 experimentos. Los resultados experimentales se reflejan en la tabla 4.1.

Vamos a explicar brevemente cada uno de los experimentos que se realizaron. Para todos ellos se empleó una misma traza de 50.000 paquetes de un tamaño de 7Mb.

**1º** El primer experimento consistió en guardar cada uno de los paquetes en un documento diferente.

**2º** El segundo experimento generaba un documento por cada 10.000 paquetes con el guardado masivo de datos y empleando el objeto BSON.

**3º** En el tercer experimento cambiamos el objeto BSON por BCON

**4º** Por último se aplicó la técnica de actualizar un documento cada vez que quisiéramos guardar un paquete.

Los tiempos obtenidos de los experimentos anteriores, que han quedado reflejados en la tabla 4.1, son las medias de realizar varias veces la misma operación para analizar el comportamiento del experimento.

**Tabla 4.1 Resultados experimentales sobre el indexado de datos**

| 1º Experimento | 2º Experimento | 3º Experimento | 4º Experimento |
|---|---|---|---|
| 1 s | 0.348 s | 0.4 s | 5 s |

Podemos ver un extracto del código con las diferencias más representativas que se emplearon para cada uno de los experimentos, en los anexos del 1 al 4.

A la vista de los resultados experimentales, se determinó que la técnica de guardado más eficiente para nuestro proyecto era la combinación del guardado

masivo de datos, junto con el objeto BSON.

En los siguientes puntos explicaremos, qué estructura hemos empleado en la base de datos para guardar los datos de los paquetes y cual es el número óptimo de paquetes por cada documento.

## 4.2 Estructura de los datos

El siguiente paso que vamos a explicar es el tipo de estructura que se decidió para guardar los datos de los paquetes dentro de la base de datos.

Para guardar los paquetes en la base de datos necesitamos una estructura definida de tipo clave->valor, para poder generar más adelante un algoritmo para poder recoger la información y poder mostrarla al usuario.

Por ello hemos definido la siguiente estructura dentro de la base de datos:

_id[mS,mD,Tmp,iS,iD,fT,eT,pS,pD,prot,off, size][ms,mD,Tmp...]...,

Como vemos, la estructura no es más que un conjunto de arrays que contienen la información que nos interesa de cada uno de los paquetes y todos ellos pertenecen al documento que se identifica con la variable _id.

Ahora vamos a explicar que son cada uno de estos parámetros:

_id = nombre del paquete + 1 número; para aclarar el _id de los paquetes, imaginemos que una trama dentro de la base de datos se divide en 20 trozos de 90000 paquetes, entonces tendríamos _id=paquete1, _id=paquete2, así uno por cada trozo de trama que guardemos en la base de datos.

mS = Mac origen.

mD = Mac destino.

Tmp = Timestamp.

iD = IP destino.

iS = IP origen.

eT = Ether type.

pS = Puerto origen.

pD = puerto destino.

prot = tipo de protocolo IP.

off = Posición binaria del paquete respecto de la traza.

size = Este campo sólo lo contiene el primer documento de cada archivo donde se almacenan los primeros paquetes. Este número marca en cuantos documentos está dividido el archivo.

## 4.3 Script packet details

El código de programación que se encarga de recoger la información que queremos guardar en la base de datos está basado en una librería creada por el profesor Daniel Morato escrita en el lenguaje de programación C. Se le han añadido modificaciones para que sea capaz de devolver en qué posición binaria se encuentra el paquete que hemos leído respecto del programa original y la parte que controla el almacenamiento en la base de datos con la API de MongoDb, para el lenguaje de programación C que hemos visto en los puntos anteriores.

Este script analiza cada uno de los paquetes que contiene el archivo, saca la información que queremos guardar en la base de datos y va generando los objetos BSON para luego abrir un cliente con el método de guardado de datos masivos y guardar los documentos.

## 4.4 Mejora del rendimiento

Una vez que conocemos como funciona la parte de nuestro proyecto que guarda los datos, vamos a explicar cómo se ha mejorado el rendimiento de todo lo visto anteriormente, para acercar los tiempos de guardado de información a la escritura en disco sobre texto plano.

Uno de los aspectos importantes a la hora de mejorar el rendimiento para guardar datos es el número de paquetes que puede insertar en un mismo documento.

Se realizaron varios experimentos reduciendo el número de paquetes de 10.000 hasta 10. Al hacer esto nos acercamos al caso del experimento número 1 en el que guardamos un paquete por cada documento y como quedó demostrado experimentalmente, esto perjudica al rendimiento.

El siguiente paso experimental fue aumentar el número de paquetes y se comprobó que no se producía ningún cambio en los resultados obtenidos.

En la siguiente tabla vemos los resultados obtenidos:

**Tabla 4.2 Resultados de las variaciones de paquetes.**

| 10 paq. | 100 paq. | 1.000 paq. | 40.000 paq. | 15.000 paq. | 20.000 paq. |
|---|---|---|---|---|---|
| 0.90 s | 0.78 s | 0.70 s | 0.392 s | 0.378 s | 0.38 s |

La conclusión que sacamos de los resultados experimentales fue que a partir de 10.000 paquetes en un mismo documento no varía el tiempo que tarda insertarlos.

El otro aspecto importante para mejorar el rendimiento del programa fue el número de clientes que insertan datos al mismo tiempo, es decir, controlar por

hilos [19] los clientes que guardan la información en la base de datos.

Para ello asignamos a cada usuario un hilo independiente con recursos independientes entre ellos.

Para comparar el rendimiento que habíamos conseguido, comparamos el guardado de los datos con una traza de 1Gb, con hilos y sin hilos.

En la tabla observamos la mejora del rendimiento cuando implementamos los hilos.

**Tabla 4.3 Resultado del rendimiento con hilos.**

| 1Gb Con hilos | 1 Gb Sin hilos |
|---|---|
| 45 Segundos | 1 minuto 40 segundos |

# 5. Aplicación web

En este apartado vamos a explicar cómo funciona y qué tecnologías implementa la parte en la que el usuario puede examinar una trama a través de la aplicación web que hemos desarrollado.

La aplicación web se alojará en un servidor local que esté en la empresa, dentro del servidor será donde coloquemos las tramas que queremos analizar, ya sea el servidor la misma sonda, o podamos enviar la trama que hemos capturado en la sonda al servidor por medio de la red.

## 5.1 Menú inicial

La aplicación web arranca en un menú de inicio, en el que se muestran los archivos que tienes ya guardados, y te da la posibilidad de añadir uno nuevo o borrar uno existente.

En la siguiente figura presentamos una imágen del menú inicial.

**Figura 5.1 Menú inicial de la aplicación web.**


Cuando pulsamos sobre el nombre del archivo que queremos abrir, nos lleva a la siguiente pantalla.

Para indexar una traza nueva, solo hay que pulsar en el botón que nos dice “Indexar traza”. Una vez lo pulsamos, nos aparece un formulario que nos permite seleccionar una traza que contenga nuestro servidor.

Entonces, ejecuta una llamada a un script PHP que guarda el nombre del archivo que hemos seleccionado en un fichero de texto, para luego poder recuperar el nombre de los archivos rápidamente, y, además, ejecuta una instrucción en la máquina del propio servidor para abrir el programa que se encarga de guardar la información necesaria en la base de datos, como hemos explicado en el punto 4.

De tal manera que quedan indexados todos los paquetes en la base de datos.

## 5.2 Visualizador de paquetes

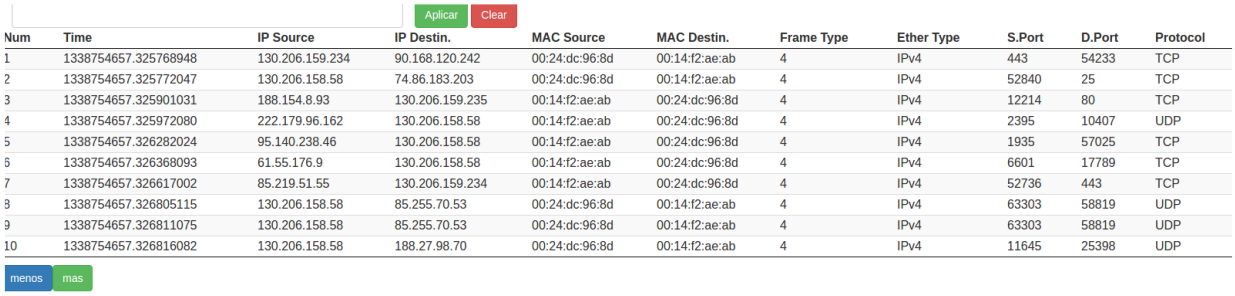
Una vez que tenemos indexados los paquetes en la base de datos, nos

aparecerá en el menú inicial el nombre de todos los archivos que se hayan indexado.

Bastará con hacer click en cualquiera de ellos para que nos muestre una tabla en la que se visualizan los paquetes del archivo de 10 en 10.

La figura nos muestra dicha tabla:

**Figura 5.2 Visualización de los paquetes**




Para visualizar los paquetes, lo que hace la aplicación web es llamar a una serie de programas escritos en el lenguaje de programación PHP que, mediante la API de MongoDB para el lenguaje de programación PHP, recoge los 20 primeros paquetes de la base de datos correspondientes a esa trama.

En el Anexo 5 se explica un extracto del código que permite realizar búsquedas y devolver la información al usuario.

Cuando el script escrito en el lenguaje de programación en PHP devuelve al cliente la información de los paquetes que se encuentran en la base de datos, actualiza la tabla dinámicamente mediante el framework dataTable de jquery.

Para acceder a la información que contiene el paquete basta con hacer click sobre uno de los paquetes. En ese momento una función AJAX hace una llamada a un script PHP enviándole la posición binaria del paquete que queremos analizar en el archivo.

Posteriormente, cuando el PHP recibe la posición binaria en la que se encuentra el paquete, manda analizar, mediante tshark, el paquete y recibe la respuesta en

formato XML que, por último, devuelve al cliente para que lo interprete y muestre al usuario como se muestra en la figura 5.3.

**Figura 5.3 Detalles del paquete**

geninfo
Number Hex(1)
Frame Length Hex(52)
Captured Length Hex(52)
Captured Time Hex(1338754657.325901000)
frame
Interface id: 0
Encapsulation type: Ethernet (1)
Arrival Time: Jun 3, 2012 22:17:37.325901000 CEST
Time shift for this packet: 0.000000000 seconds
Epoch Time: 1338754657.325901000 seconds
Time delta from previous captured frame: 0.000000000 seconds
Time delta from previous displayed frame: 0.000000000 seconds
Time since reference or first frame: 0.000000000 seconds
Frame Number: 1
Frame Length: 82 bytes (656 bits)
Capture Length: 82 bytes (656 bits)
Frame is marked: False
Frame is ignored: False
Protocols in frame: eth:ip:tcp
eth
ip
tcp

## 5.3 Filtro de paquetes

Puede que al usuario le interese la posibilidad de buscar unos paquetes en concreto. Para ello hemos programado un filtro que busca dentro de la base de datos los campos de los paquetes que coincidan con los que hemos especificado en el filtro.

Podemos hacer búsquedas con el operador lógico AND sobre los campos de IP y MAC de los paquetes.

La estructura general de los comandos del filtro son los siguientes:

X.Y==N && X.Y==N

Donde X es el campo sobre el que queremos actuar, Y si es origen o destino y N el valor que queremos que contenga.

Por ejemplo: Si queremos mostrar todos los paquetes que tengan la IP origen 192.168.13.1 y la IP destino 192.168.23, el comando que tendríamos que escribir en el filtro sería el siguiente:

ip.src==192.168.13.1&&ip.dst==192.168.23.2

En el caso de que queramos filtrar por IP origen y MAC destino, tendríamos que poner el siguiente comando:

ip.src==192.168.13.1&&mac.dst==11:11:11:11:11:ab

Al usuario solo se le mostrarán los paquetes que cumplan dichas condiciones. Además, como en el caso de la visualización normal, cuando se realiza una búsqueda en el filtro, se realiza una carga de 10 paquetes para mostrarlos inmediatamente al usuario y de otros 10 paquetes que se precargan para que el usuario pueda visualizarlos sin tener que esperar a realizar una nueva consulta en la base de datos. De esta manera aumentamos la rapidez con la que el usuario lee los datos y da la impresión de que es una lectura en tiempo real de la base de datos.

# 6. Conclusiones

Para concluir, vamos a destacar las ventajas e inconvenientes frente al resto de analizadores de tramas así como las tasas de velocidad que hemos conseguido actualmente.

Como ventajas principales podemos destacar el ahorro en costes de procesado, ya que solo necesitamos abrir la trama una sola vez para procesarla ya que posteriormente se queda guardado en qué posiciones binarias están los paquetes que contiene. Además podemos visualizar la trama en tiempo real, ya que el tiempo de lectura de los paquetes es ínfimo.

A todo ello hay que añadir que no hay límite en el tamaño máximo de la trama que queremos indexar y, posteriormente, analizar como en otros softwares.

Otra ventaja, desde el punto de vista económico para la empresa que se dedique a distribuir este software, es que no requiere despliegue de servidores para almacenar las trazas ni alojar la herramienta.

Los principales inconvenientes son la falta de herramientas tan potentes para el análisis como el resto de softwares por ser un prototipo. Asimismo, requiere un técnico especializado para montar el software en la red.

A la vista de las ventajas e inconvenientes podemos decir que beneficia a la empresa que pueda usar la herramienta en cuanto a costes de tiempo para analizar los paquetes dada su rápida apertura de los paquetes. Incluso está pensada para que usuarios limitados puedan conectarse a la vez al servidor en el que está la trama. Esto es posible gracias a que el servidor está dentro de la propia red de la empresa, por lo que no hay que tener en cuenta un acceso masivo a servidores ajenos en los que se almacenan las trazas.

Como la herramienta está pensada para que se instale dentro de la empresa que se quiere analizar el tráfico, permite a la empresa que emplea la herramienta para ejecutar el análisis de los paquetes no disponer de servidores

propios en los que se almacenan las tramas, por lo que no tiene que establecer un límite de tamaño para las tramas, así como ahorrar costes tanto en el despliegue de servidores como en la seguridad que deberían tener para proteger los datos de los clientes.

En cambio todo ello se puede llevar a cabo si hay un técnico que se encargue de colocar una sonda en la empresa donde se pretenda llevar a cabo el análisis, además de la instalación de un servidor web con la herramienta.

Para terminar podemos decir que esta versión del analizador de tramas es solo un prototipo, al que con un desarrollo posterior se puede ir añadiendo mayores herramientas para el análisis de las tramas, mejorar el filtrado de los paquetes y, dependiendo del escenario donde nos encontremos, desplegar una base de datos Mongo DB con varias máquinas para mejorar el rendimiento.

# Referencias


[1] Wireshark Foundation. (s.f.). Documentación oficial del software. W*ireshark*. Recuperado de https://www.wireshark.org/

[2] Wikipedia. (2017). Definición del Big data. Recuperado de https://es.wikipedia.org/wiki/Big_data

[3] MongoDB, Inc. (2017). Documentación oficial. *MongoDB. Recuperado de* https://www.mongodb.com/

[4] Wikipedia. (2017). Definición del concepto NoSQL. Recuperado de https://es.wikipedia.org/wiki/NoSQL

[5] López Rodríguez, J. R. (2013). Definición del concepto Documental en bases de datos no relacionales. *Grao en Información e Documentación: Bases de datos documentales.* Recuperado de http://docencia.lbd.udc.es/bdd-grao/teoria/tema1/1.2-BDsDocumentales-Introduccion.pdf

[6] Wikipedia. (2017). Definición del concepto SQL. Recuperado de https://es.wikipedia.org/wiki/SQL

[7] Wikipedia. (2013). Definición del concepto BSON. Recuperado de https://es.wikipedia.org/wiki/BSON

[8] JSON. (s.f.) Manual y definición. *JSON.* Recuperado de http://www.json.org/json-es.html

[9] MIT; ERCIM; Keio; Beihang. (2006). Definición de HTML. *W3C.* Recuperado de https://www.w3.org/html/

[10] Java Script. (2016). Manual y definición. *Java Script.* Recuperado de https://www.javascript.com/

[11] MIT; ERCIM; Keio; Beihang. (2006). Definición de CSS y su documentación ofical. *W3C.* Recuperado de https://www.w3.org/standards/webdesign/htmlcss

[12] The PHP Group. (2001). Definición de PHP y su documentación oficial. *PHP.* Recuperado de http://php.net/manual/es/intro-whatis.php

[13] The jQuery Foundation. (2017). Documentación oficial. *Ajax.* Recuperado de http://api.jquery.com/jquery.ajax/

[14] Wikipedia. (2017). Definición de Framework. Recuperado de https://es.wikipedia.org/wiki/Framework

[15] Otto, M. y Fatrick. (s.f.). Documentación oficial. *Bootstrap.* Recuperado de http://getbootstrap.com/

[16] SpryMedia Ltd. (2007). Documentación oficial. *DataTables.* Recuperado de https://datatables.net/

[17] Cloud Shark. (2016). Documentación oficial. *Cloud Shark.* Recuperado de https://www.cloudshark.org/

[18] Wikipedia. (2017). Definición de API. Recuperado de https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones

[19] Chuidiang. (2007). Definición de hilo y ejemplos de uso. *Ejemplos de java y C/linux.* Recuperado de *http://www.chuidiang.org/clinux/procesos/procesoshilos.php*

[20] Libbson 1.6.3. (2007). Documentación oficial de la librería bcon.h. *Libbson.* Recuperado de http://mongoc.org/libbson/current/index.html

[21] GitHub, Inc. (2017). Documentación oficial del driver C para Mongo Db. *MongocDB.* . Recuperado de https://github.com/mongodb/mongo-c-driver/blob/master/tests/test-mongoc-collection.c

[22] MongocDB C Driver 1.6.3. (s.f). Documentación oficial. *MongocDB.* Recuperado de http://mongoc.org/libmongoc/current/tutorial.html

# Anexos

## Anexo 1: Código del 1º experimento para insertar paquetes en MongoDB.

Recordamos que el primer experimento de inserción de paquetes fue aquel en el que guardamos un documento en la base de datos por cada uno de los paquetes que leíamos.

En el siguiente extracto de código podemos observar el proceso de cómo se realizaba la operación de guardar un nuevo documento.

```
 int i = 0;
   unsigned char   *macOutput = (unsigned char*)macInput;
   char query[100];
   char aux[17];
   char datosMongo[17];
   char opcionC[2];
   strcpy( query, "./insertar " );
   sprintf(opcionC,"%i",opcion);
   strcat(query,opcionC); //de esta manera ya tenemos la opcion que le queremos pasar a
mongo para que guarde una cosa u otra.
   strcat(query," ");
   for (i = 0; i < 5; i++) {
      if (macOutput[i] < 16)
            {
                  if(i==0)
                  {
                    sprintf(datosMongo,"0%x",macOutput[i]&0xFF); //guardamos el
hexadecimal en el string de los datos que queremos guardar

                  }else
                  {
                    strcat(datosMongo,":");
                    sprintf(aux,"0%x",macOutput[i]&0xFF);
                    strcat(datosMongo,aux);
                  }

            }else{
                  if(i==0)
```

```
                    {
                      sprintf(datosMongo,"%x",macOutput[i]&0xFF); //guardamos el
hexadecimal en el string de los datos que queremos guardar

                    }else
                    {
                      strcat(datosMongo,":");
                      sprintf(aux,"%x",macOutput[i]&0xFF);
                      strcat(datosMongo,aux);
                    }

                }
                //ahora tengo que hacer una variable que concatene todas las posiciones de la
mac en una misma para guardarla en mongodb

    }

  //if (macOutput[i] < 16) printf("0%x", macOutput[i]&0xFF);
  //else printf("la mac es:%x", macOutput[i]&0xFF);
  strcat( query, datosMongo);//concatenamos la query que queremos hacer al sistema con
el string que nos han pasado
  system(query); //ejecutamos la query en la shell del sistema para llamar a  programa
externo que guarde en mongodb.
```

Este extracto de código de la primera versión del programa que se encarga de indexar cada paquete como un único documento en la base de datos, el funcionamiento es el siguiente:

En esta primera versión, el script que se encarga de coger los datos e insertarlos está separado del código en C, se emplea directamente comandos del sistema para dar órdenes al cliente de la base de datos, por eso la última línea ejecuta un system, y el comando lo genera con la información que recoge de cada uno de los paquetes.

Esto se debe a que se realizaron pruebas primarias sin implementar la API de MongoDb para el lenguaje de programación C.

# Anexo 2: Código del 2º experimento para insertar paquetes en MongoDB.

Antes de analizar el código recordamos que el 2º experimento para indexar paquetes en la base de datos lo realizamos mediante la API de MongoDb, junto con los objetos BSON que nos proporciona la librería bson.h.
Para guardar el documento empleamos el método bullk, que corresponde al guardado masivo de datos dentro de la base de datos.

```
if(contador ==0)
		{
			bulk = mongoc_collection_create_bulk_operation (collection, true,
NULL);
			bson_init (&doc);
			bson_append_document_begin (&doc, aux3,-1, &child);
			BSON_APPEND_UTF8(&child,"macSrc",macS);
			BSON_APPEND_UTF8(&child,"macDst",macD);
			BSON_APPEND_UTF8(&child,"Timestamp",Timestamp);
			BSON_APPEND_UTF8(&child,"ipSrc",ipSrc);
			BSON_APPEND_UTF8(&child,"ipDst",ipDst);
			BSON_APPEND_INT32(&child,"frameType",frameType);
			BSON_APPEND_INT32(&child,"etherType",etherType);
			BSON_APPEND_INT32(&child,"fameSize",frameSize);
			bson_append_document_end (&doc, &child);
		}else
		{
			bson_append_document_begin (&doc, aux3,-1, &child);
			BSON_APPEND_UTF8(&child,"macSrc",macS);
			BSON_APPEND_UTF8(&child,"macDst",macD);
			BSON_APPEND_UTF8(&child,"Timestamp",Timestamp);
			BSON_APPEND_UTF8(&child,"ipSrc",ipSrc);
			BSON_APPEND_UTF8(&child,"ipDst",ipDst);
			BSON_APPEND_INT32(&child,"frameType",frameType);
			BSON_APPEND_INT32(&child,"etherType",etherType);
			BSON_APPEND_INT32(&child,"fameSize",frameSize);
			bson_append_document_end (&doc, &child);
		}
		if(contador==10000)
		{
			mongoc_bulk_operation_insert (bulk, &doc);
			ret = mongoc_bulk_operation_execute (bulk, &reply, &error);
```

```
    if (!ret) {
            fprintf (stderr, "Error: %s\n", error.message);
           }
            bson_destroy (&doc);
           mongoc_bulk_operation_destroy (bulk);
           contador =0;
           return;
}
```

Como vemos esta parte del código es la que se encarga de generar el objeto BSON y guardarlo en la base de datos.

Al entrar en la función, el código evalúa la variable contador, que es la que determina el número de paquetes que se han leído hasta el momento. Si esta variable vale 0 significa que es el primer paquete leído de todos, por lo que tendremos que generar la cabecera del objeto BSON mediante el comando bson_init (&doc). Además, también generamos un objeto llamado bulk que prepara la base de datos para recibir una cantidad masiva de información mediante el siguiente comando:

```
bulk = mongoc_collection_create_bulk_operation (collection, true, NULL).
```

Las siguientes veces que llamemos a la función, la variable ya no será 0 si no que tendrá un valor entre 1 y 10.000.

Cuando tengamos un valor menor a 10.000, procederemos a realizar un apéndice al objeto BSON que hemos creado anteriormente. Una vez tengamos 10.000 paquetes analizados, entonces procedemos a realizar un guardado del documento que contiene 10.000 paquetes, mediante el comando insert de la API de MongoDb.

# Anexo 3: Código del 3º experimento para insertar paquetes en MongoDB.

En dicho experimento probamos a sustituir el objeto BSON por el objeto BCON, para generar las cabeceras más fácilmente, al ser un objeto de un nivel superior ya que estamos definiendo los datos que va a llevar en cabecera, requiere más tiempo de procesado y por eso es más lento.

```
if(contador ==0)
            {
            doc = BCON_NEW("macSrc",BCON_UTF8
(macS),"macDst",BCON_UTF8 (macD),"pktTime",BCON_UTF8
(Timestamp),"ipSrc",BCON_UTF8 (ipSrc),"ipDst",BCON_UTF8
(ipDst),"frameType ",BCON_INT32 (frameType),"frameSize",BCON_INT32
(frameSize),"etherType",BCON_INT32(etherType));
   bson_oid_init (&oid, NULL);
            }else
            {
                  BSON_APPEND_UTF8(doc,"macSrc",macS);
                  BSON_APPEND_UTF8(doc,"macDst",macD);
                  BSON_APPEND_UTF8(doc,"Timestamp",Timestamp);
                  BSON_APPEND_UTF8(doc,"ipSrc",ipSrc);
                  BSON_APPEND_UTF8(doc,"ipDst",ipDst);
                  BSON_APPEND_INT32(doc,"frameType",frameType);
                  BSON_APPEND_INT32(doc,"etherType",etherType);
                  BSON_APPEND_INT32(doc,"fameSize",frameSize);
            }
```

Comparando este código con el experimento anterior, vemos como la principal diferencia es en la definición de la cabecera, ya que especificamos los parámetros que vamos a pasarle en el objeto, como si de cualquier lenguaje orientado a objetos se tratase.

# Anexo 4: Código del 4º experimento para insertar paquetes en MongoDB.

Este experimento trataba de actualizar un documento en vez de crear múltiples documentos al mismo tiempo, teóricamente se reduce el tiempo en cuanto a crear un documento nuevo cada vez, ya que lo que le cuesta a la base de datos tiempo es crear los índices e insertar un nuevo documento.
Pero en la práctica no es cierto ya que para actualizar un documento, la operación que realiza la base de datos a nivel de memoria es copiar todo el documento en otra parte de memoria, registrar el cambio y guardarlo de nuevo.
Por eso los resultados fueron los peores de todos.

Como ejemplo del experimento pondremos solo el comando que hay que emplear para actualizar el documento en vez de guardar uno nuevo, como hemos visto anteriormente mediante el método bulk.

```
mongoc_bulk_operation_update (bulk3, query, doc, true);
```

Donde bulk es el objeto bulk que vamos a emplear para mandar el documento a la base de datos, query es el objeto en formato BSON, como hemos explicado anteriormente; doc es el objeto del documento y true es para indicarle que tiene que ser una operación síncrona.

## Anexo 5: Código de la consulta a la base de datos

Se explica, a continuación, un fragmento del código que se emplea para seleccionar paquetes en MongoDB.

Lo primero de todo como en cualquier base de datos es realizar una conexión con la misma. En este caso para conectarnos mediante PHP a la base de datos de MongoDB emplearemos las funciones que vienen en la API de MongoDB para PHP.

```
$conexion = new MongoClient();

$bd = $conexion->test;

$coleccion = $bd->test;
```

Como vamos a emplearlo en otros scripts de PHP, hemos generado un archivo único para la conexión llamado conexion.php, que incluiremos en otros scripts de PHP cuando se requiera una conexión de este tipo.

A continuación se explica un ejemplo de código para seleccionar únicamente paquetes sin ningún tipo de filtrado ni operaciones.

```
    $archivo ="1.pcap";

    $documento =$archivo.$i;

    $query=array("_id"=>$documento);

    $campos = array("size"=>1);

    $cursor=$coleccion->find($query,$campos);

    $array2 = iterator_to_array($cursor);

    $size = $array2[$documento]["size"];
```

Lo que hacemos con este código es preparar el documento que vamos a leer y los campos que queremos consultar del mismo. En este caso queremos consultar el campo size del documento 1.pcap, que corresponde al archivo 1.pcap.

Para recuperar la información de un paquete hay que realizar la consulta de acuerdo a la estructura que se ha planteado en el apartado 4 para indexar un paquete en la base de datos.

```
$mS = $j."."."mS";

$mD= $j."."."mD";

$Tmp = $j."."."Tmp";

$iS = $j."."."iS";

$iD = $j."."."iD";

$fT = $j."."."fT";

$eT = $j."."."eT";

$pS = $j."."."pS";

$pD=$j."."."pD";

$prot = $j."."."prot";

$campos
=array($mS=>1,$mD=>1,$Tmp=>1,$iS=>1,$iD=>1,$fT=>1,$eT=>1,$pS=>1,$pD
=>1,$prot=>1);

$cursor = $coleccion->find($query,$campos);

$array = iterator_to_array($cursor);
```

*$array[$documento][$j]["eT"]=mapear($array[$documento][$j]["eT"]);*

*$narray=array("Num"=>$j,"Tmp"=>$array[$documento][$j]["Tmp"],"iS"=>$array[$documento][$j]["iS"] ,"iD"=>$array[$documento][$j]["iD"] ,"mS"=>$array[$documento][$j]["mS"] , "mD"=>$array[$documento][$j]["mD"] , "fT"=> $array[$documento][$j]["fT"],"eT"=>$array[$documento][$j]["eT"],"pS"=>$array[$documento][$j]["pS"],"pD"=>$array[$documento][$j]["pD"],"prot"=>$array[$documento][$j]["prot"]);*

*$narray["prot"] = mapeoProtocolo($narray["prot"]);*

*$Garray[] = $narray;*

Este código realizar una petición a la base de datos, para buscar un paquete que contenga esas claves, que corresponden a los datos indexados de un paquete en concreto.

La consulta se devuelve en el cursor y lo vamos guardando en un array que codificaremos en JSON para devolverlo a la aplicación cliente e interprete los resultados, para mostrarlos al usuario.

Al usuario le muestra la información de los 10 primeros paquetes, y los otros 10 los muestra cuando pulsamos los botones de más o menos para ir navegando por la trama, de tal manera que va precargando siempre 10 paquetes para mostrarlos inmediatamente cuando pasamos de página.