UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

DESARROLLO DE UN SISTEMA DE MONITORIZACIÓN DE RED BASADO EN TECNOLOGÍAS DE TRATAMIENTO MASIVO DE DATOS

Mario González Valero

Tutor: Lluís Gifre Renom

Ponente: Jorge Enrique López de Vergara Méndez

JUNIO 2018

DESARROLLO DE UN SISTEMA DE MONITORIZACIÓN DE RED BASADO EN TECNOLOGÍAS DE TRATAMIENTO MASIVO DE DATOS

AUTOR: Mario González Valero

TUTOR: Lluís Gifre Renom

PONENTE: Jorge Enrique López de Vergara Méndez

High Performance Computing and Networking Research Group (HPCN)

Dpto. Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio de 2018

Resumen

Internet es una herramienta fundamental en el día a día de la sociedad moderna, tanto en el sector empresarial como personal. Está presente prácticamente en todos los campos de conocimiento y desarrollo de la sociedad, por lo que ha empezado a resultar necesario tener un control sobre los datos que están siendo transportados por la red. De esta forma, y debido a que Internet hoy en día es un servicio que llega a millones de usuarios, es requisito indispensable el poder analizar parámetros como la latencia de la red, su congestión o, en definitiva, el estado de la red. El objetivo final es ofrecer un servicio estable y tratar de garantizar que se cumplen los acuerdos de servicio entre operadores de red y clientes.

Anteriormente, los sistemas de monitorización empleaban interfaces basadas en texto para mostrar información de los paquetes circulando por la red. Sin embargo, esta forma de operar puede conllevar dificultades a la hora de descubrir posibles problemas y pueden originar largos periodos de análisis para localizar su origen. Por este motivo, disponer de sistemas de monitorización capaces de representar visualmente el comportamiento de la red se ha convertido en una necesidad. Estos sistemas permiten identificar de forma mucho más rápida y clara, entre otros, el origen de los problemas, reduciendo así los tiempos de resolución de los mismos.

En este Trabajo Fin de Grado, se plantea diseñar y desarrollar un sistema basado en técnicas de tratamiento masivo de datos que permitan realizar una monitorización y análisis de red de la forma más visual posible. Para desarrollar el sistema planteado, primero se han analizado distintas plataformas disponibles que pueden dar soporte al sistema. Las plataformas analizadas han sido aquellas que están siendo desarrolladas y mantenidas de forma activa, lo cual aportará estabilidad y longevidad al sistema. Entre ellas, se encuentran:

- Tshark/Wireshark para la captura y disección de los paquetes. Deberemos realizar, además, una selección de los campos que resulten de interés para nuestro sistema.
- Motores de búsqueda como serán Elasticsearch, InfluxDB o Graphite.
- Para cumplir con uno de los objetivos principales de este trabajo, que sea un sistema muy visual, se analizarán plataformas como Kibana o Grafana. Estas plataformas permiten diseñar y presentar cuadros de mandos de forma fácil y ágil.

Seguidamente, se han seleccionado las plataformas y componentes que mejor se ajustan a las necesidades del sistema y finalmente, se han integrado para implementar el sistema completo.

Palabras clave

Captura de tráfico, análisis de red, monitorización de tráfico, tratamiento masivo de datos, motor de búsqueda, cuadro de mando.

Abstract (English)

Internet is a key tool in the daily life of modern society, either for business or personal use. It takes part nearly in all areas of knowledge and development of society, so it is becoming necessary to have a control over the transported data in the computers network. Thus, given that, at the present time, Internet is a service that reaches millions of users, it is mandatory to be able to analyse parameters such as network latency, congestion or, in short, the network state. The final objective is to provide a stable service and ensure the fulfilment of service agreements between network operators and customers.

Previously, monitoring systems used text-based interfaces in order to present the information related to packets transported on the network. However, this way of working can originate difficulties when it is necessary to discover possible problems and could imply long periods of analysis to find the origin of the problems. For this very reason, it is of paramount importance to have monitoring systems able to visually represent the network performance. These systems enable, among others, fast localization of the origin of problems, thus reducing their resolution times.

This end-of-degree thesis aims to design and develop a Big Data-based system to perform network monitoring and analysis of the collected data as visually as possible. To develop the proposed system, as an initial step, different platforms and frameworks currently available that can support the system have been analysed. The tools considered in this analysis are those that are being actively developed and supported, thus providing stability and longevity to the system. In particular, the analysed tools are:

- Tshark / Wireshark for packet capture and traffic dissection. In fact, we have to select those fields that will be important for our system.
- Search engines, such as Elasticsearch, InfluxDB or Graphite.
- To achieve one of the main objectives of this work, i.e. to build a visual system, platforms such as Kibana or Grafana will be analysed. These platforms enable to design and of present dashboards in an easy and agile manner.

Afterward, those platforms and components that fit in the best way with the system's needs have been selected; finally, they have been integrated to implement the complete system.

Keywords

Traffic capture, network analysis, traffic monitoring, Big Data, search engine, dashboard.

Agradecimientos

Llega el momento de poner fin a una etapa muy importante de mi vida. Algunos años de sacrificio que, sin duda, han merecido la pena. Una etapa que comenzaba con la idea de estudiar una carrera que se ajustaba a una de mis pasiones que es la telefonía móvil, aunque, en general, bastante apasionado de cualquier tecnología revolucionaria que pueda llegar a transformar el mundo y la forma en la que vivimos y pensamos.

Quisiera agradecer:

A mis padres y a mi hermano, por aguantarme durante todo este tiempo, en mis alegrías, en mis tristezas, en mis momentos de nervios, en momentos difíciles y, en general, por estar ahí siempre. Por animarme a seguir adelante, a no rendirme nunca y a seguir trabajando con el objetivo bien claro. Gracias infinitas, porque desde luego, sin vosotros no habría sido posible.

Por supuesto, a mi tutor y ponente, Lluís Gifre y Jorge E. López de Vergara. Un privilegio haber podido disfrutar de dos personas con tan amplios conocimientos, dispuestos a enseñar y transmitir todo el conocimiento posible para ayudarme a realizar este Trabajo Fin de Grado, pero también porque son muy buenas personas y se han portado de una manera fantástica conmigo. Muchas gracias.

A mis compañeros, en especial a Juan y Luis, por hacerlo todo más ameno, por las risas, por el buen ambiente y por animarme todo este tiempo. Gracias.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Glosario

API	Application Programming Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
CNAME	Canonical Name
CSV	Comma-Separated Values
DNS	Domain Name System
ELK	Elasticsearch Logstash Kibana
ES	Elasticsearch
GB	Gigabyte
Gbps	Gigabit por segundo
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
MAC	Media Access Control
MB	Megabyte
MIT	Massachusetts Institute of Technology
MPLS	Multiprotocol Label Switching
PB	Petabyte
PCAP	Packet Capture Application Programming
RPM	Red Hat Package Manager
SQL	Structured Query Language
TB	Terabyte
TCP	Transmission Control Protocol
TICK	Telegraf InfluxDB Cronograf Kapacitor
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
XML	Extensible Markup Language

1 Introducción

1.1 Motivación

Desde su aparición en el año 1969, Internet ha experimentado un gran desarrollo. Lo que en un primer momento nació como un proyecto para conectar las universidades de los Estados Unidos entre sí, así como algunas instituciones, conocido con el nombre de ARPANET, hoy en día es una herramienta clave en el desarrollo de la sociedad, tanto para uso personal, como para el desarrollo empresarial.

Según un estudio realizado por Cisco Systems [1], se estima que durante el año 2018, unos 150.910 PetaBytes (PB, 10^{15} bytes) de información han cruzado Internet teniendo en cuenta a todos los continentes. Sin embargo, este mismo estudio supone que para el año 2021, la cifra incrementará hasta los 278.108 PB al año, o lo que es lo mismo, en tan solo 3 años, el tráfico será casi el doble del tráfico actual teniendo en cuenta cualquier tráfico IP que cruza Internet.

El tráfico en Internet ha aumentado debido al incremento de usuarios, pero también porque se han mejorado los servicios suministrados. Hablamos de nuevos servicios que han aparecido en los últimos años, como el vídeo bajo demanda, la realidad virtual, las tiendas online, etc. Así como otros que surgieron en los inicios de Internet y que han evolucionado desde entonces, como es el caso del correo electrónico. Este último, cuando apareció, simplemente permitía enviar mensajes en texto plano, sin embargo, hoy en día, un correo electrónico puede incorporar una gran cantidad de datos incluyendo imágenes, vídeos y archivos adjuntos de cualquier tipo. Es evidente que ahora se requiere una mayor cantidad de datos para soportar dichos servicios.

Otro tipo de aplicaciones que está apareciendo en la actualidad es el llamado Internet of Things (IoT), en el que dispositivos cotidianos como una nevera pueden tener acceso a Internet para realizar pedidos de la compra automatizados, o podemos encontrar sensores, por ejemplo meteorológicos, distribuidos geográficamente que envían periódicamente datos a algún servicio en la nube. El disponer de dispositivos que necesitan conectarse, aunque sea de manera intermitente, a Internet, hace que aumente el tráfico, ya no sólo por el tiempo de conexión, sino por la gran cantidad de conexiones que originarán dichos dispositivos en los próximos años.

Todo lo anterior ha provocado que haya un aumento considerable de los datos circulando a través de Internet. Esto ha venido provocado por el desarrollo de la tecnología y la consecuente reducción del precio de los dispositivos. En la década de los 60, almacenar unos pocos de MegaBytes (MB, 10^6 bytes) podía suponer unas máquinas de un tamaño considerable mientras que a día de hoy, cualquier teléfono inteligente o unidad de almacenamiento posee capacidades de almacenamiento de varios GigaBytes (GB, 10^9 bytes) por un precio proporcionalmente mucho menor y en un tamaño mucho más reducido.

Este desarrollo tecnológico ha fomentado que los ordenadores sean, proporcionalmente, más rápidos, baratos y, por tanto, accesibles a más gente. Esto viene de la mano con una población que sigue aumentando con el paso de los años, lo que ha permitido que cada vez

el número de personas que tiene acceso a dispositivos conectados a Internet sea mayor, aumentando directamente el tráfico de datos en Internet.

De igual forma, la tasa de transmisión de los datos también ha aumentado significativamente. Actualmente, muchas de las comunicaciones se realizan a través de cables de fibra óptica, los cuales permiten unas tasas de transmisión de datos y un alcance muy superiores a las que se tenían años atrás con los cables eléctricos. Los cables de fibra óptica han contribuido a mejorar las comunicaciones entre los continentes, formando así toda una red de cables transoceánicos submarinos. Por poner un ejemplo, un solo enlace de 100 Gigabits por segundo (Gbps), es capaz de transmitir del orden de 1 PB de datos por día. Asumiendo paquetes Ethernet de longitud 1500 bytes, resulta en unos 720 millones de paquetes por día. Podemos ver con este ejemplo que para disponer de un sistema de análisis que nos permita estudiar el estado de una red realista durante varios días o semanas, la cantidad de datos a procesar es muy elevada y motiva la necesidad de aplicar modernos sistemas y técnicas de tratamiento masivo de datos para poder realizar un correcto análisis del tráfico de la red.

1.2 Objetivos

Como consecuencia de lo expuesto anteriormente, el objetivo principal de este Trabajo Fin de Grado es desarrollar un sistema de monitorización de red basado en técnicas de tratamiento masivo de datos que permitan analizar de manera visual el tráfico ayudando al usuario a comprender qué está sucediendo en su red. Para ello, se han definido una serie de sub-objetivos que permitan desarrollar con éxito el sistema para analizar el tráfico de red; de esta manera, se creará una variedad de "cuadros de mandos" con gráficos en la que se verá representada toda la información de los paquetes. Entre estos sub-objetivos, se pueden destacar los siguientes:

- **Captura de tráfico**
  - Será necesario capturar el tráfico de los equipos de red monitorizados, bien capturando directamente los paquetes, o bien tomando los datos desde fuentes externas que los almacenen en ficheros para su posterior análisis, como puedan ser ficheros PCAP (Packet Capture Application Programming, programación de aplicaciones de captura de paquetes). En este sub-objetivo estudiaremos el proceso de captura del tráfico de red.
- **Limpieza y tratamiento previo de los datos**
  - Una vez capturados los datos, es necesario extraer la información relevante de las capturas. Se pueden obtener una gran cantidad de datos, correspondientes a los distintos protocolos. Sin embargo, sólo algunos de estos campos serán de utilidad para el análisis. En este sub-objetivo extraeremos de los paquetes los datos relevantes para el sistema de monitorización para su posterior análisis.
- **Indexación de los datos**
  - Para poder acceder a toda la información y, posteriormente, poder representarla de manera gráfica, será necesario almacenar e indexar los datos en un motor de búsqueda. En este sub-objetivo analizaremos distintos motores de búsqueda disponibles y seleccionaremos el que mejor se ajuste a las necesidades del sistema.
- **Representación de los datos**
  - Por último, la información indexada deberá poder ser representada de forma gráfica para el usuario haciendo uso de plataformas de representación analítica de datos. Esto nos permitirá conseguir una visualización sencilla y

clara del estado de la red. En este sub-objetivo analizaremos distintos plataformas disponibles y seleccionaremos la que mejor se ajuste a las necesidades del sistema.

1.3 Fases de realización del proyecto

Durante la realización de este Trabajo Fin de Grado, se han dividido las horas de desarrollo en varias fases que han permitido completar con éxito el proyecto planteado en un inicio. En el siguiente diagrama Gantt de la Figura 1-1, se puede ver de manera clara la división por semanas de las tareas realizadas. En la Tabla 1-1 aparecen detalladas las tareas realizadas.

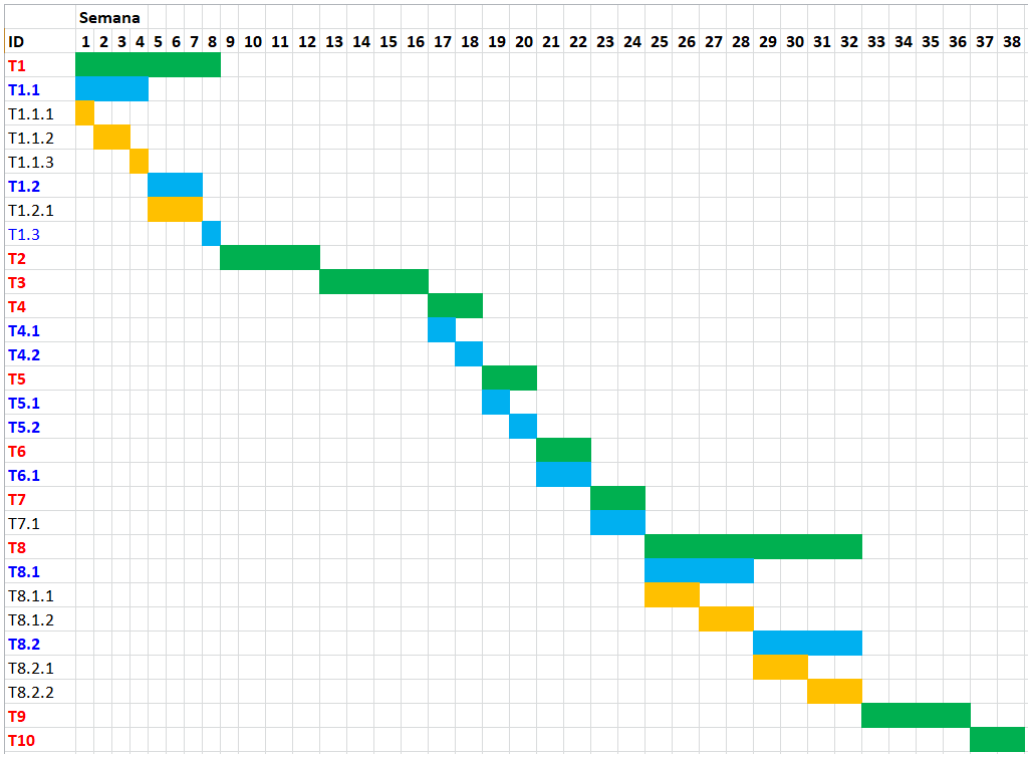

Figura 1-1 – Diagrama de Gantt de desarrollo del trabajo

Tabla 1-1 – Listado de tareas

Tarea	Nombre
T1	Estudio de la pila ELK
T1.1	Estudio de Elasticsearch
T1.1.1	Estudio de JSON
T1.1.2	Estudio de los mappings de Elasticsearch
T1.1.3	Indexación de datos en Elasticsearch
T1.2	Estudio de Logstash
T1.2.1	Elaboración de ficheros de indexación
T1.3	Estudio de Kibana
T2	Estudio de InfluxDB
T2.1	Indexación de datos en InfluxDB
T3	Estudio de Graphite
T3.1	Indexación de datos en Graphite
T4	Estudio de Grafana

T4.1	Representación de datos
T4.2	Creación de los paneles
T5	Estudio de Tshark
T5.1	Análisis de rendimiento
T5.2	Selección de los campos de utilidad
T6	Automatización del proceso de captura e indexación
T6.1	Desarrollo de script en Bash
T7	Establecimiento de escenarios
T7.1	Estudio del Packet Analyzer de Steelcentral
T8	Desarrollo del sistema final
T8.1	Sistema de análisis de trazas
T8.1.1	Moloch
T8.1.2	Desarrollo del sistema ES + Logstash + Grafana
T8.2	Sistema de análisis en tiempo real
T9	Redacción de la memoria
T10	Elaboración de la presentación

1.4 Organización de la memoria

Con el objetivo de presentar de manera clara y detallada el proceso de realización del trabajo, así como los resultados obtenidos, la memoria se ha organizado en los siguientes capítulos:

- **Capítulo 2: Estado del arte.** En este capítulo se van a presentar los principales conceptos y tecnologías disponibles en el campo de la monitorización de redes y entornos de procesado masivo de datos que serán de relevancia para este trabajo.
- **Capítulo 3: Diseño.** En este capítulo se realizará un estudio sobre los sistemas de bases de datos y motores de búsqueda que más se están utilizando actualmente para poder elegir de una manera justificada cuál de ellos será el empleado para desarrollar el sistema de monitorización objetivo de este trabajo. De igual forma, se estudiarán las plataformas que nos permitan realizar los cuadros de mandos en los que visualizar los datos.
- **Capítulo 4: Desarrollo.** En este capítulo se expondrá cómo se ha realizado el proceso de captura de tráfico, los campos relevantes que se han seleccionado y cómo estos se van a utilizar para diseñar los distintos escenarios de aplicación del sistema que se empleen para la validación de este último.
- **Capítulo 5: Integración, pruebas y resultados.** Una vez desarrollado nuestro sistema, se presentarán una serie de comparativas con otras soluciones similares a las desarrolladas para encontrar las ventajas e inconvenientes de cada sistema.
- **Capítulo 6: Conclusiones y trabajo futuro.** Por último, en este capítulo se presentan las conclusiones y posibles líneas de investigación futuras que se derivan de este trabajo.

2 Estado del arte

En este capítulo, se va a hacer un breve repaso al estado actual de las tecnologías relevantes para este trabajo y se estudiará el por qué resulta de interés utilizar nuevas técnicas para la monitorización del tráfico de red.

2.1 Captura de tráfico

Para poder conocer el estado de una red, es necesario obtener una captura de los paquetes que están circulando a través de ella, pudiendo así, analizar diversos parámetros que permitan extraer una conclusión sobre su estado. Para ello se deben utilizar herramientas de captura de tráfico de red. Entre todas ellas, una de las más conocidas, maduras y referenciadas es Wireshark / Tshark.

La herramienta Wireshark, originariamente conocida como Ethereal, surgió a finales del año 1997 [2] como un sistema que permitiera a su autor, Gerald Combs, revisar el estado de sus redes. Dicho sistema resultaba realmente útil para la comunidad y, desde que comenzó el proyecto, fue muy bien recibido y muchas personas contribuyeron a su desarrollo, mayoritariamente creando disectores para los distintos protocolos de red. En el año 2006, se le cambió el nombre a Wireshark (nombre actual de la herramienta) y en el año 2008 se presentó Wireshark versión 1.0, considerada la primera versión completa y totalmente funcional. En el año 2015 se lanzó la versión 2.0 con una nueva interfaz gráfica de usuario.

Por otra parte, se encuentra Tshark que es la versión en línea de comandos de Wireshark. Tshark resulta de gran interés cuando es necesario utilizar la herramienta en un entorno que no dispone de interfaz gráfica, o simplemente, cuando es necesario automatizar procesos. En las Figura 2-1 y Figura 2-2 se presenta, respectivamente, ejemplos de captura de pantalla de Tshark y Wireshark.

```
Capturing on 'wlan0'
    1 0.000000000 104.155.32.164 →172.16.180.205 TLSv1.2 324 New Session Ticket, Change Cipher Spec, Encrypted Handshake
    2 0.002453964 172.16.180.205 →104.155.32.164 TLSv1.2 622 Application Data 2018-04-19 14:16:45,997412346
    3 0.002963273 18.184.58.214 →172.16.180.205 TLSv1.2 1823 Application Data 2018-04-19 14:16:45,997921655
    4 0.002984816 172.16.180.205 →18.184.58.214 TCP 66 44514 →443 [ACK] Seq=1 Ack=1758 Win=352 Len=0 TSval=1002900482 T
    5 0.002997544 52.212.62.134 →172.16.180.205 TCP 74 443 →58548 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1386 SACK_
    6 0.003014990 172.16.180.205 →52.212.62.134 TCP 66 58548 →443 [ACK] Seq=1 Ack=1 Win=229 Len=0 TSval=3439933289 TSec
    7 0.003526502 54.36.212.192 →172.16.180.205 TLSv1.2 559 Application Data 2018-04-19 14:16:45,998484884
    8 0.005607044 172.16.180.205 →52.212.62.134 TLSv1 583 Client Hello 2018-04-19 14:16:46,000565426
    9 0.006330449 18.194.102.149 →172.16.180.205 TCP 74 443 →39640 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1386 SACK_
   10 0.006356755 172.16.180.205 →18.194.102.149 TCP 66 39640 →443 [ACK] Seq=1 Ack=1 Win=229 Len=0 TSval=1970227729 TSe
   11 0.007201277 172.16.180.205 →150.244.214.200 DNS 90 Standard query 0xc3d7 A ads.stickyadstv.com OPT 2018-04-19 14:1
   12 0.007402509 172.16.180.205 →18.194.102.149 TLSv1 583 Client Hello 2018-04-19 14:16:46,002360891
   13 0.007556952 172.16.180.205 →178.250.0.76 TCP 66 46142 →443 [ACK] Seq=1 Ack=1 Win=39846 Len=0 TSval=1477465049 TSe
   14 0.007673694 172.16.180.205 →150.244.214.200 DNS 87 Standard query 0x5ae6 A idsync.rlcdn.com OPT 2018-04-19 14:16:4
```

Figura 2-1 – Ejemplo de captura de paquetes en Tshark

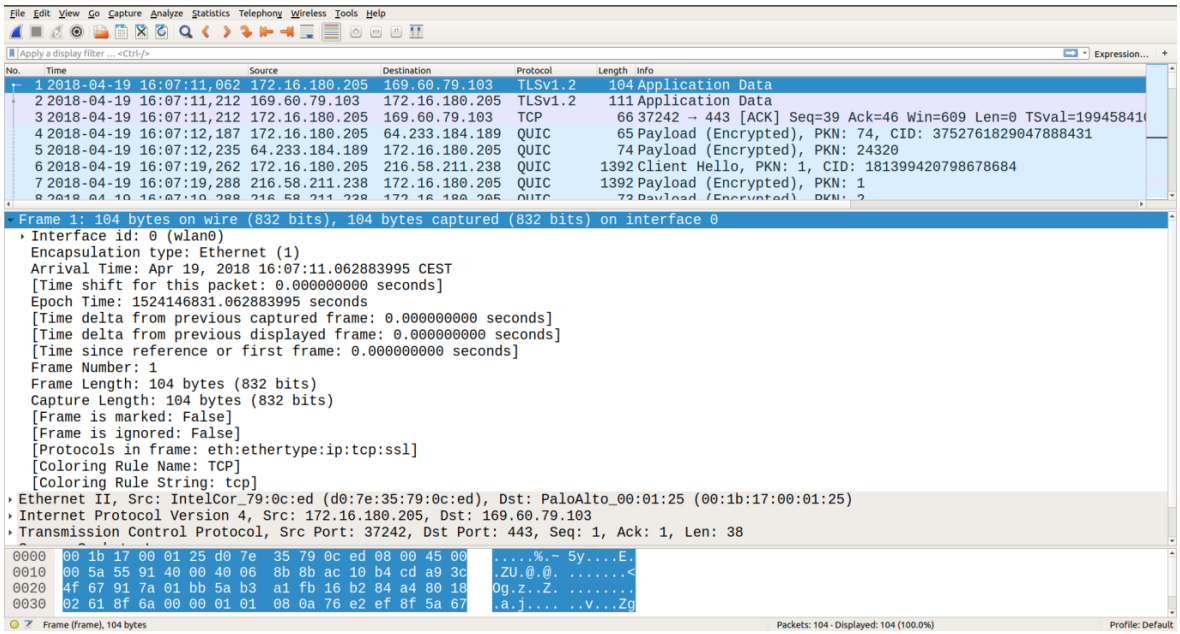

Figura 2-2 – Ejemplo de captura de paquetes en Wireshark

De una manera más concreta, Wireshark es un analizador de protocolos de red. Las funciones principales de Wireshark son, por una parte, capturar tráfico de red en tiempo real en las interfaces de red seleccionadas, o bien, leer paquetes de ficheros de traza que ya habían sido capturados con anterioridad y proceder a su análisis. Wireshark se basa en el uso de la librería libpcap para la captura de los paquetes y permite guardar las capturas en múltiples formatos, entre ellos PCAP. Además, ofrece una interfaz gráfica de usuario sobre la cual se visualizan los paquetes y se accede a todas sus funcionalidades. Wireshark está disponible tanto para Windows, como para Linux y MacOS.

Entre las principales funcionalidades de Wireshark, aparte de la propia disección de los paquetes de red, podemos encontrar la posibilidad de realizar un estudio de las conversaciones entre direcciones IP origen y destino, así como con direcciones MAC, ver las estadísticas de uso de cada protocolo o resolver direcciones IP u obtener algunos datos como su geolocalización.

Aun cuando Wireshark ofrece multitud de funcionalidades, la visualización de los datos se realiza sobre una interfaz caracterizada por un alto contenido textual, donde el uso de gráficos que permitan visualizar de una manera sencilla el estado de la red es escaso. Wireshark ofrece algunas herramientas para realizar gráficos como los gráficos de entrada/salida (I/O graphs) donde existe la posibilidad de realizar gráficas sobre ciertos parámetros de la red. Por supuesto, nada de esto está disponible en su alternativa Tshark. Además, es necesario tener un alto conocimiento de la herramienta para poder aprovechar al máximo todo su potencial.

En cualquier caso, la principal limitación de Wireshark está relacionada con sus limitaciones en el tamaño de las trazas que pueden ser analizadas. Cuando se capturan o cargan trazas de paquetes de unos cientos de MB o incluso algún GB; cosa muy normal hoy en día en cualquier empresa de tamaño medio/grande, los tiempos de procesado de Wireshark incrementan notablemente, llegando al punto de introducir tiempos de procesado inaceptables en la resolución de problemas.

Aun cuando Wireshark no resulta efectiva como herramienta de análisis para trazas suficientemente grandes, su versión en línea de comandos Tshark, en este caso, en su

versión 2.4.6, formará parte fundamental en el desarrollo de este proyecto para poder automatizar las capturas y exportaciones de paquetes, en combinación con guiones de comandos en Bash.

2.2 Procesado de datos masivo

Debido al aumento de la cantidad de datos que circulan por Internet, ha surgido la necesidad de utilizar nuevas técnicas para poder analizar volúmenes ingentes de información en tiempos razonables. Hace unos años apareció el término procesamiento masivo de datos, normalmente denominado como "*Big Data*" en inglés, para referirse a este nuevo paradigma que se origina cuando las herramientas utilizadas tradicionalmente para analizar y procesar los datos quedan obsoletas por no satisfacer un tiempo de procesado razonable. Mientras que en la década de los 80 era impensable hablar de tamaños de datos del orden de los TeraBytes, hoy en día, se pueden instalar en un ordenador personal discos duros con varios TeraBytes de capacidad de almacenamiento. Además, el desarrollo de nuevas aplicaciones y servicios como las plataformas de video on-line y las redes sociales, generan un aumento significativo de la cantidad de datos que producen los propios usuarios y que deben ser transmitidos, procesados y almacenados en los servidores distribuidos por Internet.

El paradigma Big Data se puede resumir en 3 características fundamentales sobre los datos conocidas como "las tres V's":

- **Volumen:** la cantidad de datos que se generan es demasiado grande como para ser procesados y/o almacenados utilizando técnicas clásicas.
- **Velocidad:** los datos se generan a una velocidad que impide a un sistema clásico procesarlos en tiempo real.
- **Variedad:** los datos pueden ser de tipos muy variados (imágenes, videos, textos, audios, etc) complicando su procesado empleando técnicas clásicas.

Posteriormente, surgió una cuarta "V":

- **Veracidad:** los datos tienen que tener su origen en fuentes fiables y de calidad para así poder generar resultados también de calidad.

Con el objetivo de poder procesar la información para estudiarla, han aparecido diversos componentes y módulos *software* que operan bajo el paradigma Big Data. Estos componentes y módulos pueden combinarse entre ellos formando paquetes integrados. Típicamente, estos paquetes integrados acostumbran a incorporar un motor de búsqueda, un recolector de datos y una plataforma de visualización, aunque pueden incorporar componentes adicionales según la aplicación para la que hayan sido diseñados. Así tenemos paquetes, como la pila ELK (Elasticsearch, Logstash, Kibana Stack), la pila TICK (Telegraf, InfluxDB, Chronograf, Kapacitor Stack) o Graphite, que están desarrollados con el fin de satisfacer las exigencias de tratar con grandes cantidades de datos. Dichas herramientas serán estudiadas en capítulos posteriores de este trabajo.

La capacidad de estas nuevas herramientas para tratar grandes cantidades de datos, supone que, al menos a priori, la idea de utilizar herramientas Big Data para realizar análisis de red puede suponer un avance importante en este ámbito.

2.3 Herramientas similares

En este capítulo se van a estudiar una serie de herramientas cuya finalidad es similar a la que se plantea como objetivo de este trabajo para poner en contexto nuestro sistema planteado con algunas soluciones ya existentes.

2.3.1 Steelcentral Packet Analyzer

Desarrollado por la empresa norteamericana Riverbed [3] con sede en San Francisco, California, esta herramienta es un capturador y analizador de paquetes de red cuyo objetivo es ofrecer una interfaz gráfica muy visual al usuario para identificar de una manera más rápida y sencilla cualquier irregularidad en la red.

Se trata de una herramienta de pago desarrollada para sistemas Windows. Permite analizar las interfaces de red del sistema sobre el que se está ejecutando, así como ficheros de traza PCAP. Packet Analyzer ofrece integración con Wireshark para un análisis más profundo del tráfico capturado. De hecho, Riverbed es patrocinador de Wireshark, por tanto, existe una estrecha relación entre ambos.

Packet Analyzer ofrece una división en apartados de distintos tipos de análisis de la red, distinguiendo así entre análisis IP, análisis LAN, análisis de la red inalámbrica, etc. Además de esto, por cada uno de esos apartados, se ofrecen distintas visualizaciones de los datos, entre los que se pueden encontrar: gráficos temporales, diagramas de flujo, histogramas o diagramas tipo tarta; en los que se resume el análisis del fichero PCAP cargado.

En la Figura 2-3, se puede ver un ejemplo de análisis de una traza PCAP donde se está representando el top de IPs utilizadas en forma de histograma.

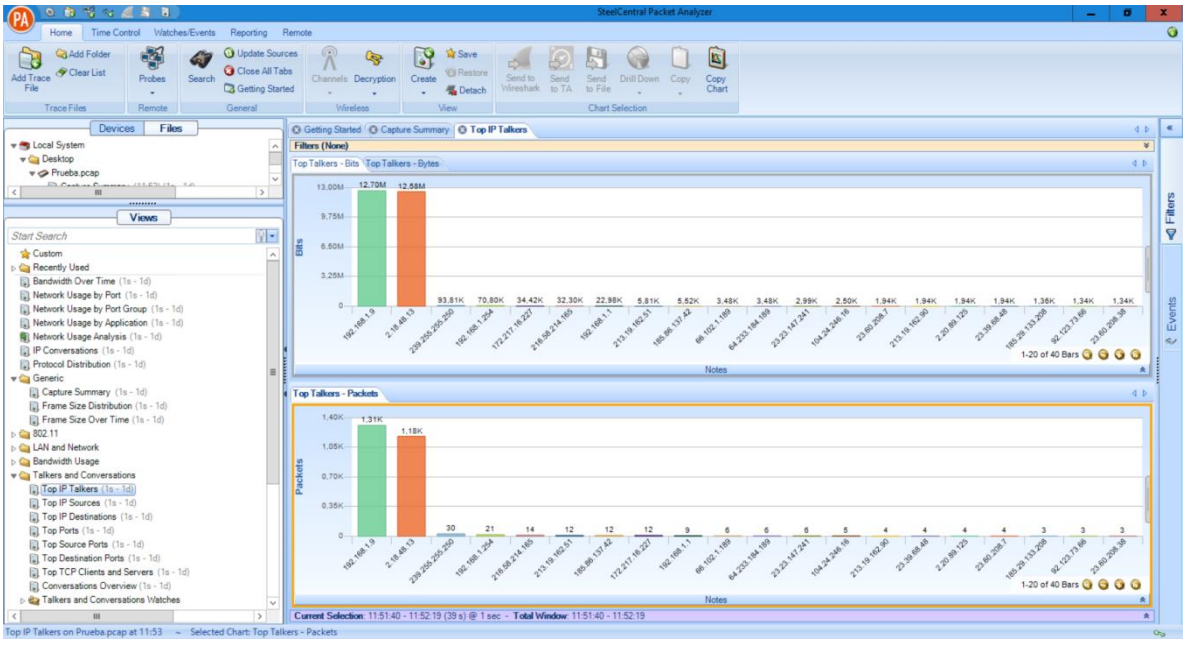

Figura 2-3 – Ejemplo de visualización en Packet Analyzer

El sistema está desarrollado para ser capaz de tratar grandes cantidades de datos y, además, estos pueden estar repartidos en periodos distintos, desde unas horas hasta días, semanas o incluso meses de capturas completas.

Packet Analyzer de Riverbed es una herramienta similar a la que se plantea en el desarrollo de este TFG. Sin embargo, la diferencia fundamental reside en que Packet Analyzer es una herramienta de pago, mientras que la que se propone en este TFG está desarrollada utilizando sistemas gratuitos de acceso libre.

2.3.2 Moloch

Moloch es un sistema escalable de captura de paquetes e indexación de los datos desarrollado por AOL [4]. Utiliza como motor de indexación Elasticsearch y permite realizar un análisis de los paquetes capturados de una manera visual y gráfica. Permite, tanto el análisis de paquetes en tiempo real, como la indexación de paquetes previamente capturados en formato PCAP de momentos anteriores para, posteriormente, ser cargados y estudiados.

Según el equipo de desarrollo de Moloch, está basado sobre tres pilares fundamentales alrededor de los cuales gira todo el sistema. Estos son: la seguridad, la escalabilidad y la interfaz.

En cuanto a la seguridad, el sistema emplea un canal seguro sobre protocolo HTTPS (HyperText Transfer Protocol Secure / protocolo seguro de transferencia de hipertexto) para comunicar al usuario con el sistema y permite también el uso de contraseñas. Posteriormente, todos los ficheros capturados son almacenados en los servidores de Moloch, sólo pudiendo acceder a ellos por medio del uso de la interfaz gráfica de Moloch o mediante su API (Application Programming Interface, Interfaz de programación de aplicaciones). Además de esto, se ofrece la posibilidad de encriptar los ficheros PCAP que no están siendo analizados en ese momento para asegurar la protección de los datos.

Otro de sus pilares fundamentales tiene que ver con la escalabilidad. Debido a que Moloch emplea nodos de Elasticsearch, se dota al sistema de dicha escalabilidad. Los desarrolladores de Moloch aseguran incluso la posibilidad de manejar varios gigabits por segundo de tráfico.

Por último, respecto a su interfaz, esta supone un punto diferenciador respecto a otros sistemas. Se pueden capturar los paquetes utilizando otro tipo de herramientas como pueda ser Wireshark y, posteriormente, utilizar la interfaz de Moloch que puede ser más intuitiva y sencilla, para realizar el estudio de las trazas de red y análisis de sesiones. Esta interfaz se ofrece como servicio web.

Aun cuando Moloch está pensado también para ser un sistema visual y amigable para el usuario, su interfaz gráfica tiene limitaciones que se pretenden subsanar con el sistema objeto de este Trabajo Fin de Grado. En cualquier caso, Moloch es una herramienta con la que comparar el sistema diseñado. En el capítulo 5, se podrán apreciar las limitaciones de visualización de Moloch comparado con el sistema desarrollado.

2.3.3 Packetbeat

Packetbeat es una herramienta de monitorización de tráfico desarrollada por el equipo de Elastic que propone capturar paquetes de la red y enviarlos a Logstash o, directamente, a Elasticsearch. Packetbeat forma parte de la pila ELK, por lo tanto, ofrece total integración con el resto de elementos de la pila. Se puede utilizar Logstash para procesar los datos antes de ser indexados en Elasticsearch añadiendo información o, directamente, indexarlos en Elasticsearch para visualizarlos en Kibana.

Packetbeat forma parte de lo que Elastic denomina la plataforma Beats. La plataforma Beats está constituida por, Packetbeat, para el tráfico de red, Filebeat, para logs, Metricbeat

para métricas y Winlogbeat para eventos de Windows. Además, Elastic da la posibilidad de crear nuevos "Beats" si se requiere realizar alguna tarea determinada.

En la Figura 2-4 se puede ver el diagrama de funcionamiento de Packetbeat:

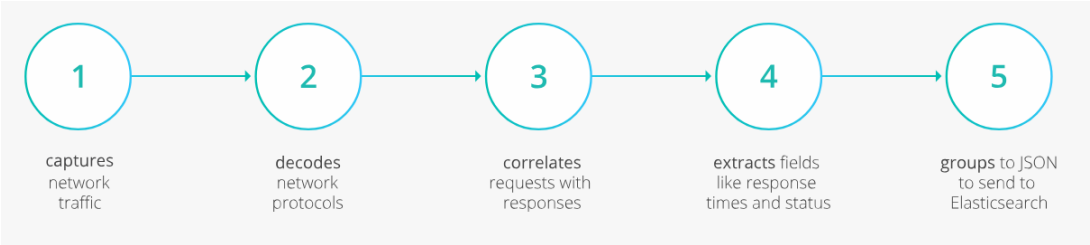

Figura 2-4 – Diagrama de funcionamiento Packetbeat (fuente [5])

2.4 Conclusiones

Debido al aumento del tráfico de red en los últimos años, los métodos tradicionales de análisis de tráfico han quedado obsoletos requiriendo de sistemas de procesamiento masivo de datos y técnicas de *Big Data* para procesar las capturas.

En este capítulo, hemos repasado el estado del arte relacionado con este trabajo. En particular, se ha hecho un breve repaso a las herramientas utilizadas para capturar tráfico de red, como son el caso de Wireshark o su variante, Tshark. Por otra parte, se ha hecho una breve introducción al paradigma Big Data, presentando sus conceptos más importantes. Por último, se han analizados herramientas similares al sistema que se plantea desarrollar en este trabajo y se ha comentado su funcionamiento.

Algunos de los sistemas introducidos en esta sección ya hacen uso de dichas técnicas de Big Data para mejorar su servicio. Sin embargo, ninguna de ellas propone la posibilidad de tener un sistema gratuito y con una alta flexibilidad en la visualización y recolección de datos, permitiendo múltiples gráficos distintos en un mismo panel, pudiendo así, analizar el estado de la red rápidamente. Este será el objeto de desarrollo de este Trabajo Fin de Grado.

3 Diseño

El objetivo fundamental de este capítulo será el de analizar y seleccionar los componentes que más se adecuen al sistema propuesto. Para ello, se hará un recorrido desde el sistema para realizar las capturas de paquetes, así como los paquetes integrados que puedan dar soporte al sistema que estarán formados por un motor de búsqueda, un sistema recolector de datos y una plataforma para la visualización de los datos.

De cada uno de estos paquetes integrados, se analizarán las características de su motor de búsqueda, el recolector y el cuadro de mandos para poder decidir finalmente cuáles formarán parte del sistema objeto de este trabajo.

3.1 Análisis de la herramienta de captura

Aunque existen varias herramientas disponibles para capturar tráfico de red, Wireshark se ha convertido en la herramienta *de-facto* para este tipo de tareas. Ofreciendo una gran cantidad de opciones para el análisis, como puedan ser la aplicación de filtros o la posibilidad de analizar qué conversaciones entre direcciones IP o direcciones MAC están teniendo lugar. Además, existe Tshark, que es la versión de Wireshark para línea de comandos y que permite automatizar tareas de captura mediante su invocación a través de guiones de comandos, por ejemplo escritos en Bash [6].

Para la realización de este trabajo, será necesario automatizar el proceso de captura para que, periódicamente, el tráfico sea capturado e indexado en un motor de búsqueda que permita finalmente estudiar el estado de la red. Es aquí donde Tshark, combinado con un guion de comandos en Bash permitirá visualizar la llegada de paquetes en el cuadro de mandos mientras estos están siendo capturados. El análisis se actualizará directamente en tiempo real en el cuadro de mandos al ir indexándose los paquetes recibidos en el motor de búsqueda.

Wireshark y Tshark son herramientas que están teniendo un desarrollo y soporte activo desde hace años, tanto para sus versiones Windows, como para Linux y MAC. Por todo lo anterior, y dado que es la herramienta *de-facto* para estas tareas, ha sido la opción seleccionada para desarrollar el sistema objetivo de este trabajo.

3.2 Análisis de paquetes integrados

Con el objetivo de poder almacenar todos los paquetes capturados para su análisis, resultará necesario estudiar diversas plataformas que dispongan, como mínimo, de un recolector, un motor de búsqueda y un cuadro de mandos. Las herramientas estudiadas son plataformas que actualmente están en auge y se actualizan periódicamente resolviendo problemas y errores que reportan los usuarios, así como introduciendo nuevas funcionalidades. Esto asegurará obtener un sistema longevo y estable sobre el que desarrollar el sistema de monitorización de red, ofreciendo garantías de correcto funcionamiento a lo largo del tiempo.

Los tres paquetes integrados que se evaluarán serán: *i*) la pila ELK, *ii*) la pila TICK y, por último, *iii*) Graphite. De todas ellas, se analizarán sus distintos componentes, motores de búsqueda, recolectores y cuadros de mandos, para seleccionar los elementos más convenientes para desarrollar el sistema de monitorización.

3.2.1 Pila ELK

Creado por el equipo de desarrollo de Elastic, la pila ELK consiste en la combinación de Elasticsearch, Logstash y Kibana [7]. El nombre proviene de la concatenación de las siglas de estos componentes. La estructura de la pila ELK se resume en la Figura 3-1, donde Logstash recibe los datos, los procesa y los carga en Elasticsearch, para, por último, ser representados con Kibana:

Figura 3-1 – Arquitectura de la pila ELK

Elasticsearch [8] es un motor de búsqueda que está basado en la API Lucene. Apache Lucene es una API que permite añadir capacidades de búsqueda a una aplicación [9]. Elasticsearch fue desarrollado por Shay Banon el cual ya tenía un sistema parecido que desarrolló en el año 2004 con el nombre de Compass. Debido a la imposibilidad de seguir desarrollando Compass de una manera sencilla ya que, según él mismo explicó, le habría resultado necesario reescribir prácticamente de nuevo todo el código para hacerlo compatible con nuevas versiones de Lucene, decidió comenzar de nuevo con la creación de un sistema que fuera capaz de lograr una alta escalabilidad.

Con esto, surge Elasticsearch que es un motor de búsqueda distribuido que usa una arquitectura basada en el principio RESTful API y desarrollado sobre Java. El principio detrás de REST API es el de establecer una interfaz entre sistemas que mediante el uso de HTTP obtenga datos o que indique que se están realizando operaciones sobre ellos [10].

La primera versión de Elasticsearch (v0.4), data de febrero del año 2010; desde entonces, ha permanecido en constante desarrollo [11]. A fecha de redacción de este TFG (mayo de 2018), Elasticsearch se encuentra en la versión 6.2.4, pasando así por más de 30 versiones diferentes, lo cual muestra claramente que el desarrollo por parte del equipo de Elastic es muy activo y vaticina longevidad para dicho componente.

Por lo planteado anteriormente, Elasticsearch supone numerosas ventajas; entre ellas, cabe destacar: *i*) estar desarrollado en Java le otorga una gran compatibilidad; *ii*) al ser un sistema distribuido, es altamente escalable; y *iii*) debido al uso de mensajes codificados en JSON [12] para las preguntas y respuestas, lo hace compatible con distintos lenguajes de programación. Además, el uso de una API RESTful permite utilizar algún cliente web de línea de comandos, como cURL [13], para subir las muestras a Elasticsearch de forma fácil. Logstash es una herramienta orientada sobre todo al tratamiento de trazas de aplicaciones. Logstash se encarga de recolectar información siguiendo una secuencia de pasos bien definida y guarda los datos en un motor de búsqueda especificado; por defecto emplea Elasticsearch, pero puede utilizar cualquier otro sistema que funcione de manera similar.

El diagrama de trabajo de Logstash se puede dividir en tres pasos fundamentales que se pueden visualizar en la Figura 3-2, donde recibe información proveniente de distintas fuentes que serán los datos que van a ser procesados en una segunda etapa para, finalmente, indexar los datos en un motor de búsqueda.

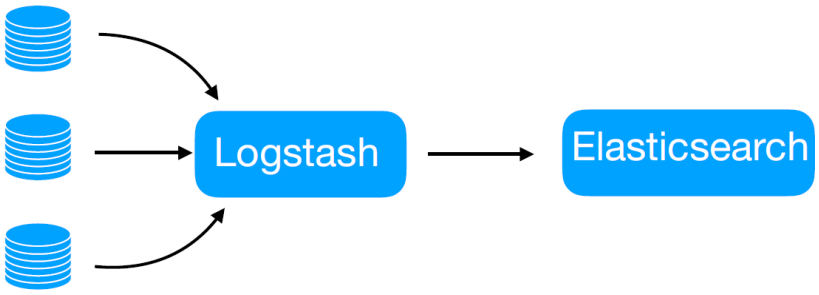

Figura 3-2 – Diagrama de trabajo de Logstash

Explicados de forma más detallada, serían los siguientes:

- **Ingesta de datos:** permite recibir datos desde diversas fuentes: desde un servidor, desde aplicaciones web, ficheros en formato CSV, etc.
- **Análisis gramatical y aplicación de filtros:** una vez recibidos los datos, Logstash posee la capacidad de procesar dichos datos y adecuarlos al formato del sistema donde serán almacenados. Esto permite realizar tareas como estructurar la información, por ejemplo, traducir registros de fichero en formato CSV a un formato JSON válido para su almacenamiento en Elasticsearch. Además, Logstash puede aplicar técnicas de geolocalización de direcciones IP o anonimizar datos de carácter personal.
- **Almacenamiento de datos:** finalmente, una vez procesados los datos, Logstash puede almacenarlos en Elasticsearch u otros destinos como CloudWatch de Amazon Web Services (AWS), InfluxDB, Graphite, etc.

El último elemento de la pila ELK es Kibana, el cual permite la representación gráfica de los datos almacenados en un índice de Elasticsearch. Kibana ofrece varias posibilidades en cuanto a la representación de los datos como puedan ser: representación de series temporales, histogramas, grafos, etc. Además, Kibana ofrece la posibilidad de extender sus funcionalidades mediante paquetes adicionales.

3.2.2 Pila TICK

En la Figura 3-3 se muestra a modo de resumen, el funcionamiento de la pila TICK que irá siendo explicado a lo largo de la sección.

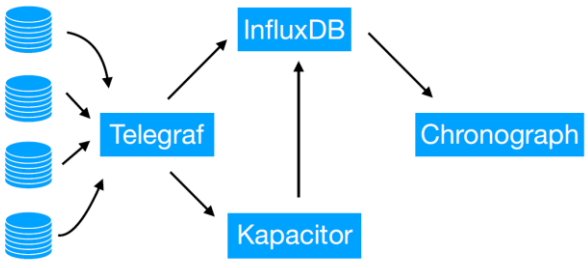

Figura 3-3 – Arquitectura de la pila TICK

Al igual que Elastic, InfluxData también ha desarrollado lo que denominan pila TICK, la cual está formada por los componentes: Telegraf, InfluxDB, Chronograf y Kapacitor [14].

El objetivo de Telegraf es el de actuar como recolector obteniendo información sobre el sistema en el que está siendo ejecutado. Una vez que se han obtenido estas métricas, vuelca su información sobre una base de datos, normalmente InfluxDB. De igual manera, permite cambiar su salida por otros sistemas de búsqueda como pueda ser Graphite. Posee la capacidad de instalar una gran cantidad de paquetes de extensión que permiten ampliar sus funcionalidades.

InfluxDB es una base de datos orientada a series temporales desarrollada por InfluxData. InfluxDB nació como proyecto open source y fue presentado en septiembre del año 2013. El objetivo principal de InfluxDB es el de ser una base de datos para grandes cantidades de información, pero cuyos datos se encuentren marcados por una marca de tiempo y pensada para recolectar directamente los datos de la máquina en la que esté siendo ejecutada para tener un rápido sistema de monitorización.

InfluxDB está escrito en Go [15], desarrollado por Google y similar a C tanto en sintaxis como en eficiencia, pero orientado a objetos; por este motivo, resulta adecuado para desarrollar un sistema complejo como InfluxDB, donde el rendimiento es un principio fundamental.

Según el equipo de desarrollo de la pila TICK, entre las características más destacables de InfluxDB podríamos encontrar las siguientes:

- **Alto rendimiento:** InfluxDB está orientado a almacenar series temporales y se consiguen obtener unas prestaciones en cuanto a ingesta de datos y tratamiento de las consultas muy elevadas.
- **Consultas similares a SQL:** InfluxData ha desarrollado un lenguaje de consultas denominado InfluxQL para interaccionar con InfluxDB de tal forma que resulte muy familiar a aquellos que ya han trabajado con SQL, reduciendo el tiempo de desarrollo de nuevas aplicaciones.

Chronograf ofrece una interfaz que permite realizar la representación de los datos almacenados en un motor de búsqueda como InfluxDB. Permite la creación de cuadros de mandos de manera sencilla, ofreciendo plantillas que pueden ser utilizadas para una rápida observación de los datos. También ofrece la posibilidad de desarrollar nuevas plantillas si fuera necesario.

El último elemento de la pila, es Kapacitor; un motor de procesamiento de datos que permite, entre otras cosas, establecer alertas gracias al uso de umbrales, con lo que se podrán detectar anomalías en los datos

En el caso particular de este TFG, se ha probado a realizar un sistema de monitorización usando InfluxDB. Con el fin de cargar estos datos a la base de datos InfluxDB, se pasó del PCAP a un CSV que contenía la misma información, pero en un formato que ocupa menos y que contiene sólo los campos cuyo análisis es de interés.

Para ello, se recurrió al uso del módulo csv-to-influxdb [16]. Este módulo, también escrito en Go, lee el fichero CSV línea a línea y carga cada línea en una base de datos creada usando InfluxDB.

Sin embargo, debido a que este módulo sólo estaba preparado para recibir la marca de tiempo de los registros en formato fecha contenida en una cadena de texto y el fichero CSV con los datos provenientes de una captura usando Tshark emplea un formato en coma

flotante, el procesado de los registros fallaba. InfluxDB convertía la marca de tiempo de los paquetes en un campo más y, por defecto, añadía una nueva marca de tiempo en un formato válido a cada entrada correspondiente al instante de inserción en InfluxDB, no al instante de captura de los paquetes.

Es por ello que fue necesario modificar el módulo csv-to-influxdb para que realizara la conversión de las marcas de tiempo de coma flotante a fecha codificada en una cadena de texto. Lo que se hizo fue comprobar si el timestamp era de tipo coma flotante, como se obtiene de Tshark; en dichas situaciones, se convertía la marca de tiempo a un formato de fecha contenida en cadena de texto que resolvió el problema de procesado. Se puede ver en el anexo A: Modificación de la función csv-to-influxdb.

Finalmente, a la hora de representar los datos en el cuadro de mandos, la visualización no fue la esperada, ya que, por ejemplo, a la hora de representar la llegada de las distintas direcciones MAC sólo representaba una de ellas. Se estuvo probando y estudiando detenidamente el problema, y al no conseguir encontrar una solución, se descartó utilizar un sistema basado en la pila TICK.

3.2.3 Graphite

Graphite es una herramienta de monitorización open source bajo licencia Apache 2.0. Fue desarrollada por Chris Davis en el año 2006 con el objetivo de que fuese una herramienta de monitorización orientada hacia el ámbito empresarial, pero de un bajo coste.

Graphite está divido en tres elementos principales [17]. En la Figura 3-4 se puede ver la arquitectura de Graphite, donde se recolectan datos que son volcados en Carbon, pasan a Whisper para ser almacenados y, por último, podrían ser representados por medio de la interfaz web.

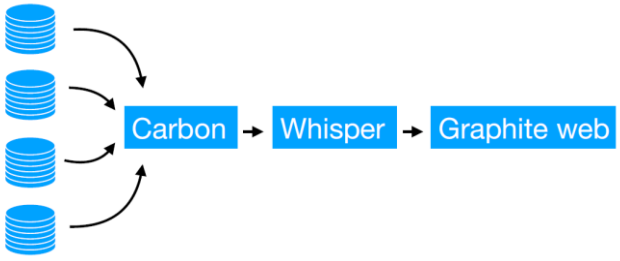

Figura 3-4 - Arquitectura de Graphite

Graphite se divide en tres componentes:

- **Carbon**: es un servicio, formado por tres procesos, cuyo objetivo es recibir datos de series temporales.
- **Whisper**: es una base de datos cuyo objetivo es el de almacenar la información que recibe en forma de series temporales. Aunque está desarrollada en Python y esto puede resultar que sea lenta, el equipo de desarrollo asegura ser “suficientemente rápida” para la mayoría de propósitos.
- **Graphite web:** es un servicio web que ofrece una interfaz de usuario en la que se representarán los gráficos resultantes de analizar los datos almacenados en Whisper.

3.3 Análisis de sistemas de cuadros de mando

En la sección anterior se han visto como cada pila incluía su propio sistema de creación de cuadros de mando, como han sido Kibana, Chronograf y Graphite web. Además de estas opciones, se analizará Grafana para poder elegir la herramienta que mejor se ajuste a nuestro sistema.

3.3.1 Grafana

Grafana es un sistema web de creación, es decir, Grafana ofrece una plantilla vacía donde se le van añadiendo distintos tipos de paneles, los cuales tendrán que ser configurados para representar finalmente los datos que se deseen. Permite una gran cantidad de representaciones distintas que van desde representaciones de series temporales, pasando por histogramas, diagramas de flujo, diagramas de tipo tarta o tablas, todos ellos personalizables en distintos aspectos. Además, para algunas de estas representaciones como, por ejemplo, las series temporales, permite establecer un sistema de alerta para detectar anomalías en los datos, aunque para ello, se necesita que el motor de búsqueda empleado o, como Grafana lo llama, *Data Source,* sea compatible con el establecimiento de alertas.

A diferencia de los sistemas de visualización de datos vistos en la sección 3.2 que sólo funcionan dentro de la pila sobre la que se hubieran integrado, Grafana ofrece la posibilidad de presentar los datos provenientes de más de 30 motores de búsqueda distintos, entre los que encontramos los ya comentados como Elasticsearch, InfluxDB y Graphite, pero también otros como Prometheus, AWS Cloud Watch, MySQL, OpenTSDB o PostgreSQL por nombrar algunos de ellos [18].

Además de esto, Grafana funciona tanto en sistemas Unix como en Mac o Windows, por lo que ofrece una gran compatibilidad. Además, Grafana emplea extensiones para aumentar sus funcionalidades, que pueden ser desarrollados por la comunidad y pueden suponer la inclusión de nuevos cuadros de mandos o de nuevas fuentes de datos.

3.4 Elección de componentes para el sistema

Una vez analizadas todas las herramientas disponibles para realizar nuestro sistema de monitorización de red, podemos decidir cuáles de ellas se ajustan mejor al sistema objetivo de este trabajo. Como herramienta para obtener las capturas de los paquetes, se ha elegido Wireshark / Tshark, pues es la herramienta *de-facto* hoy en día y porque ofrece las funcionalidades que se necesitan para desarrollar nuestro sistema.

En cuanto a las pilas analizadas, de la pila ELK nos quedamos con dos componentes: Elasticsearch y Logstash. Elasticsearch porque es un motor de búsqueda que ofrece una facilidad de uso, pero también una alta escalabilidad y permite almacenar los ficheros de capturas de paquetes de una manera sencilla. En cuanto a Logstash, es una herramienta útil para poder tratar los ficheros de captura para su posterior indexación en Elasticsearch.

En cuanto a Grafana, ha sido elegido frente a los sistemas de cuadros de mandos propios de cada pila por varias razones. Grafana recibe actualizaciones constantemente. La versión utilizada para el desarrollo de este TFG es 4.6, aunque en marzo de 2018 salió la versión 5.0 que incluye algunas variaciones en la interfaz de usuario y modificaciones de características soportadas en las distintas fuentes de datos. Por el avance en el desarrollo del TFG se ha decidido desarrollar el sistema sobre la versión 4.6 para asegurar un sistema con un funcionamiento estable y correcto.

Por ser un sistema más completo que otros como Kibana o, por ejemplo, el propio de Graphite, que ofrece la posibilidad de recibir información desde una amplia variedad de motores de búsqueda distintos, y que dispone de un sistema de extensiones mediante los cuales se pueden seguir aumentando las capacidades de representación del sistema, así como por su facilidad de uso y su gran atractivo visual, se ha elegido a Grafana como el sistema a utilizar para desarrollar este Trabajo Fin de Grado.

3.5 Conclusiones

Tras haber analizado los tres paquetes integrados: la pila ELK, la pila TICK y Graphite; y ver que ninguno en su conjunto cumple con los requisitos necesarios para la realización de este trabajo, se ha decidido optar por una solución en la que se eligen los componentes más adecuados para nuestro sistema.

Una fácil utilización, la fácil carga de datos en Elasticsearch, bien utilizando la API RESTful, o bien usando la herramienta de la pila ELK Logstash, así como un buen rendimiento y escalabilidad hacen de este sistema la mejor opción para este trabajo.

De igual manera, la gran posibilidad de personalización, así como su soporte a una gran cantidad de fuentes de datos distintas, su facilidad y correcto estilo, hacen de Grafana el otro elemento para realizar el sistema de análisis de red planteado.

En conclusión, los elementos seleccionados han sido: Elasticsearch, Logstash y Grafana.

4 Desarrollo

El objetivo de este capítulo es establecer varios escenarios diferentes que nos permitan definir distintos casos de uso de análisis del estado de una red. Debido a que Wireshark contiene disectores capaces de extraer más de 227.000 campos diferentes de más de 3000 protocolos [19], es necesario elegir cuáles de todos esos campos son realmente interesantes para un análisis de red y qué protocolos en concreto interesa conocer.

Es por esto que el objetivo de este capítulo será estudiar qué campos aportan valor al análisis para después poder almacenarlos en Elasticsearch y visualizarlos en Grafana.

4.1 Campos analizados

Para el correcto análisis del estado de la red, será necesario establecer qué campos resultarán de interés para obtener la mayor cantidad de información que sea realmente útil en los cuadros de mandos. Wireshark / Tshark ofrecen la posibilidad de obtener unos 227.000 campos diferentes de los distintos protocolos que soporta pero, evidentemente, habrá que seleccionar aquellos que permitan estudiar el estado de la red de una manera clara y sencilla. La lista de campos seleccionados, se enumera en la Tabla 4-1; esta, es similar a la que Riverbed propone en Packet Analyzer, introducido en el capítulo 2.

Tabla 4-1 – Campos analizados de Wireshark / Tshark

Nombre del campo	Protocolo	Descripción
frame.time_epoch	-	Tiempo Unix de los paquetes desde el 1 de enero de 1970 en milisegundos
frame.len	-	Longitud de los paquetes en bytes
eth.src	Ethernet	Dirección MAC origen
eth.dst	Ethernet	Dirección MAC destino
eth.type	Ethernet	Tipo de trama Ethernet
vlan.id	VLAN	ID de la red de área local virtual (VLAN)
vlan.priority	VLAN	Prioridad del paquete de usuario en la red VLAN.
mpls.label	MPLS	Valor de la etiqueta MPLS
arp.src.proto_ipv4	ARP	Dirección IPv4 del emisor ARP
arp.dst.proto_ipv4	ARP	Dirección IPv4 del receptor ARP
arp.src.hw_mac	ARP	Dirección MAC del emisor ARP
arp.dst.hw_mac	ARP	Dirección MAC del emisor ARP
ip.src	IP	Dirección IP origen
ip.dst	IP	Dirección IP destino
ip.proto	IP	Número de protocolo que identifica el protocolo del nivel superior a IP
ip.ttl	IP	Tiempo de vida del datagrama IP
icmp.type	ICMP	Tipo del mensaje ICMP
tcp.srcport	TCP	Puerto TCP origen
tcp.dstport	TCP	Puerto TCP destino
tcp.window_size	TCP	Tamaño de la ventana TCP
tcp.flags.str	TCP	Banderas de TCP en formato cadena de caracteres
tcp.analysis.out_of_order	TCP	Segmentos TCP fuera de orden

tcp.analysis.lost_segment	TCP	Segmentos TCP perdidos
tcp.analysis.zero_window	TCP	Número de ocurrencias de la condición de ventana de tamaño cero de TCP
tcp.connection.fin	TCP	Bandera TCP Fin
tcp.connection.rst	TCP	Bandera TCP Reset
udp.srcport	UDP	Puerto UDP origen
udp.dstport	UDP	Puerto UDP destino
dns.cname	DNS	Nombre canónico reportado por el servidor DNS

Estos campos serán utilizados para la creación de los escenarios planteados en la sección 4.5.

4.2 Rendimiento de Tshark

Al decidir que el sistema elegido para la realización de este TFG sería uno en el que se obtuviesen las capturas de paquetes utilizando Tshark, resultaba necesario estudiar cuál era la manera más óptima de trabajar con él. Aunque este TFG no estaba orientado a conseguir un sistema de altas prestaciones, sí que, al menos, había que estudiar algunos puntos básicos para trabajar de manera eficiente. Para ello, se realizaron varias pruebas en las que se midió el tiempo necesario para procesar una captura de tráfico y obtener de ella los campos que resultasen de interés en el análisis.

La captura en formato PCAP analizada provenía de la descarga de un fichero mediante una plataforma punto a punto; de esta forma, al tener que obtener información de distintos pares, se consigue que aumente la variedad de algunos campos como puedan ser las direcciones IP, obteniendo así mayor cantidad de datos y resultando en unas medidas más fiables de rendimiento.

Con lo anterior, se utilizó Tshark para obtener los siguientes campos mediante el comando que se puede ver en la Tabla 4-2:

Tabla 4-2 - Comando de disección Tshark para JSON

```
   time tshark -e "frame.time_epoch" -e "ip.src" -e "ip.dst"
-e "ip.proto" -e "eth.src" -e "eth.dst" -e "eth.type"
-e "tcp.srcport" -e "tcp.dstport" -e "udp.srcport"
-e "udp.dstport" -T ek -r capturatorrent.pcap > performance1.json
```

Debido a que se utilizará Elasticsearch para el desarrollo del sistema y este utiliza JSON, lo más lógico podría ser obtener los datos en un fichero JSON y, posteriormente, subirlo a Elasticsearch.

De esta forma, se ejecuta Tshark sobre el fichero de captura que se desee y se obtienen los campos que incluyen un '-e' delante. Con '-T ek' se obtiene en formato JSON y, por último, se le indica qué fichero leer '-r' y se guarda todo en el fichero performance1.json.

Para generar el fichero en formato CSV habría que ejecutar el siguiente comando, en este caso se utiliza '-T fields' para obtenerlo en formato CSV como se puede ver en la Tabla 4-3:

Tabla 4-3 – Comando de disección Tshark para CSV

```
time tshark -e "frame.time_epoch" -e "ip.src" -e "ip.dst" -e "ip.proto" -e "eth.src" -e "eth.dst" -e "eth.type" -e "tcp.srcport" -e "tcp.dstport" -e "udp.srcport" -e "udp.dstport" -T fields -r capturatorrent.pcap > performance1.csv
```

Tras realizar las pruebas cinco veces, para obtener una estimación fiable, se presenta la media de los resultados obtenidos. En la Tabla 4-4 se observa el rendimiento para el caso de almacenar la información en un fichero JSON, mientras que en la Tabla 4-5 se observa para el caso de almacenarlo en un fichero CSV.

Tabla 4-4 – Rendimiento en formato JSON

Real	112.559 s	1m 52s 559 ms
User	105.9992 s	1m 45s 304ms
Sys	6.0046 s	6.0046 s

Tabla 4-5 – Rendimiento en formato CSV

Real	94.5316 s	1m 34s 5316ms
User	93.1192 s	1m 33s 1192ms
Sys	1.3252 s	1.3252s

Con esto se puede concluir que en el caso de CSV se obtienen tiempos menores, por lo tanto, ya puede suponer un motivo importante de uso. Sin embargo, el motivo fundamental tiene que ver con el tamaño del fichero resultante. Mientras que para el caso del JSON el fichero pesa 907.6 MB, el CSV pesa 259.4 MB, lo cual supone más de tres veces menos tamaño, que, a la hora de almacenar las capturas, puede resultar crucial, aunque hay que tener en cuenta que al subir los ficheros a Elasticsearch, volverán a ser convertidos a un formato JSON y volverán a ocupar lo mismo.

4.3 Tipos de paneles

Tal como se concluyó en el capítulo 3, emplearemos Grafana para construir el módulo de visualización del sistema. Gracias a la amplia variedad de gráficos o (denominados *Panels*, en Grafana) disponibles, se podrá hacer diseñar un cuadro de mandos amplio con gran variedad de datos. Seguidamente, explicamos de forma breve los distintos tipos de gráficos que serán utilizados:

- **Gráfico temporal:** sirve para representar series temporales. En nuestro caso, resultará de especial utilidad para situaciones como la representación de las llegadas de los paquetes, ordenados por su marca de tiempo. Además de mostrar la llegada de paquetes, será utilizado también para ver la disposición temporal de los paquetes de protocolos determinados, como pueda ser TCP o UDP. En la Figura

4-1 se puede observar un gráfico temporal en el que se representan las llegadas de los paquetes.

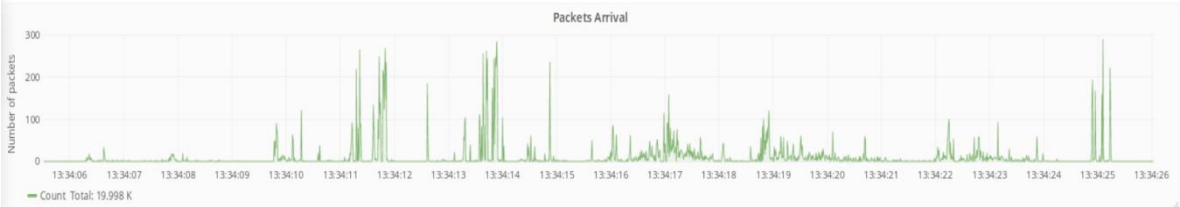

Figura 4-1 – Grafana – Graph Panel

- **Singlestat:** consiste en resumir todos los datos a un valor representativo. Esto permitirá, por ejemplo, analizar todos los paquetes y ver cuál es el de mayor o menor longitud o incluso hacer medias u otras operaciones. Además de esto, permite añadir detrás del número representativo, un gráfico que muestra la llegada de paquetes similares. En la Figura 4-2, se puede ver un ejemplo de este gráfico, donde se analiza la longitud máxima de los paquetes.

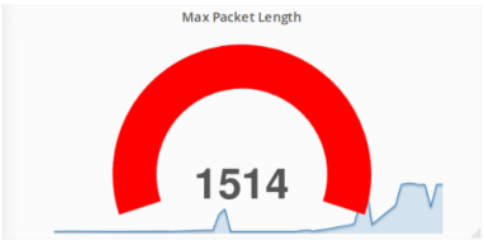

Figura 4-2 – Grafana – Singlestat Panel

- **Tablas:** permite crear tablas con capacidad de ordenación de los datos y contar su frecuencia de aparición. Esto, por ejemplo, será útil para la representación de los nombres canónicos del protocolo DNS (Domain Name System, sistema de nombres de dominios) [20]. Como se puede ver en el ejemplo de la Figura 4-3, se puede leer el nombre de dominio así como el número de veces que ha aparecido.

Figura 4-3 – Grafana – Table Panel

- **Texto:** sirve para incrustar texto fijo en el cuadro de mandos. Una funcionalidad interesante es la posibilidad de introducir el texto en formato HTML, lo cual otorga la posibilidad de incrustar texto formateado, permitiendo ajustar así el tamaño, tipo, color y disposición del texto insertado, e incluso, componentes activos como guiones de comandos o contenidos externos. En la Figura 4-4 se puede ver un ejemplo de texto formateado escrito en HTML, donde se aplican los estilos comentados a la fuente.

Talkers & Conversations

Figura 4-4 – Grafana – Text Panel

- **Mapa de calor:** funciona de manera similar a un histograma, pero a lo largo del tiempo; algo similar al gráfico temporal presentado anteriormente. En este caso, cada porción temporal en la que está dividido el gráfico representa su propio histograma, pero en vez de ser de barras, se representa con franjas que varían su tonalidad dependiendo de los datos que contenga. En la Figura 4-5 se puede ver un mapa de calor aplicado sobre las direcciones MAC de origen, donde se representa la llegada de los paquetes en el tiempo, a la vez que se hace una clasificación atendiendo a la cantidad de ocurrencias, representada en escala de verdes.

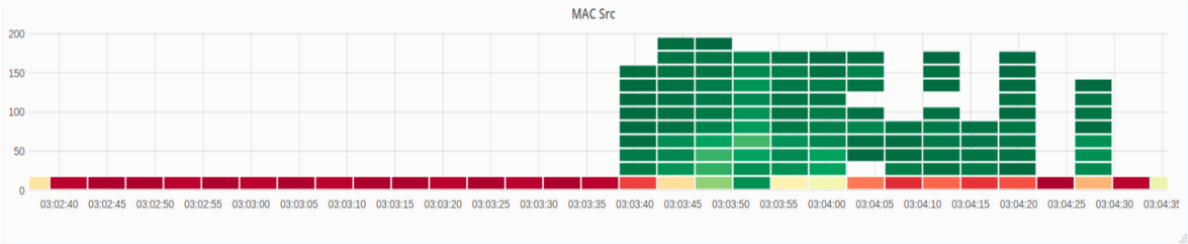

Figura 4-5 – Grafana – Heatmap Panel

- **Gráfico de tarta:** es un gráfico circular que representa un porcentaje sobre el total. Es útil, por ejemplo, para ver el porcentaje de ocurrencias de cada dirección IP, dirección MAC, etc. Este diagrama puede ayudar a detectar, por ejemplo, ocasiones en las que una sola dirección IP inunda de tráfico la red, o la distribución de los puertos TCP o UDP usados. Este gráfico no viene por defecto en Grafana y ha de ser instalado como extensión. En la Figura 4-6 se puede ver un diagrama de tipo tarta para representar las direcciones IP de origen de una captura de paquetes junto con el detalle de dichas direcciones IP en la parte inferior del gráfico.

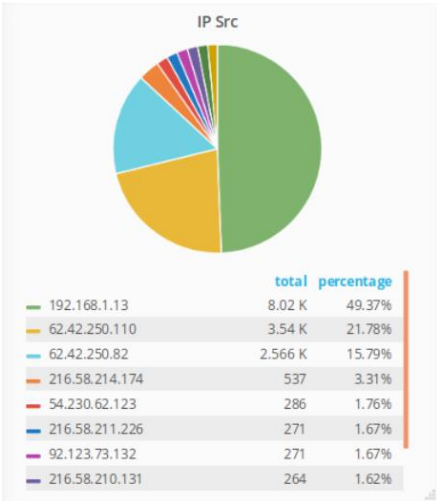

Figura 4-6 – Grafana – Pie Chart Panel

- **Diagrama:** permite crear diagramas de flujos entre los datos. Este diagrama ha de ser instalado también como una extensión. Para definir un diagrama, se debe introducir sus datos siguiendo la sintaxis de Mermaid [21].

  En la Figura 4-7 se puede observar un diagrama que relaciona las direcciones MAC que se han comunicado entre sí.

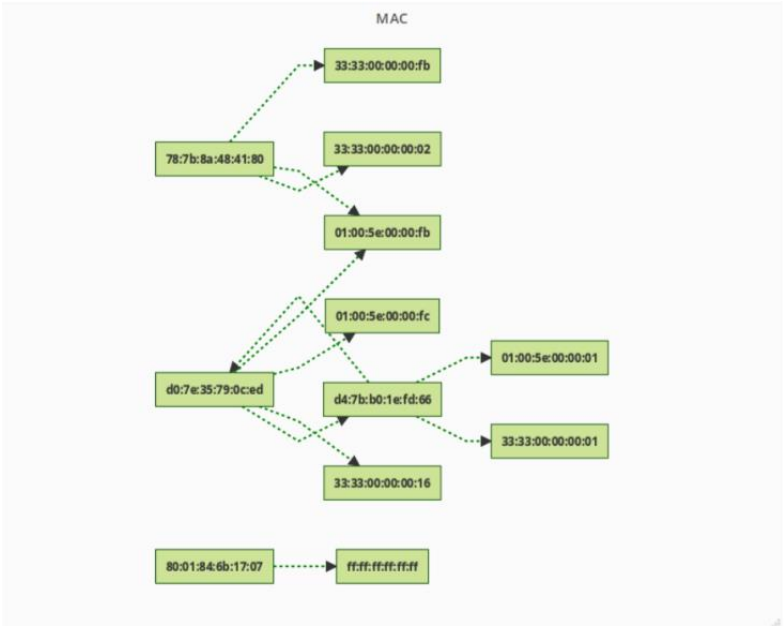

Figura 4-7 – Grafana – Diagram Panel

4.4 Generación de la sintaxis de Mermaid

Con el panel de diagrama podemos, por ejemplo, utilizarlo para representar las relaciones entre las direcciones IP de origen y destino, así como las direcciones MAC de origen y destino. Para realizarlo, ha sido necesario desarrollar un sencillo script en Python que trate el fichero CSV que contiene los datos con las direcciones IP y MAC, tanto de origen como de destino. Dichas direcciones se insertan en dos listas y, mediante el uso de '*set*', se eliminan los duplicados para que no aparezcan las líneas más veces si se repiten pares de direcciones origen y destino. Finalmente, la salida del script genera la sintaxis necesaria para introducirla en la configuración del panel y obtener el diagrama. Esto se puede ver en el anexo B: Generación de la sintaxis Mermaid.

Esta extensión también da la opción de obtener la sintaxis a través de una URL que devuelva el texto en formato Mermaid. Para ello, se realizó un servidor utilizando web.py que se encargase de realizar esto. No se continuó con este desarrollo porque Grafana tenía problemas para llamar al servidor a través de la URL, por lo que el gráfico no se actualizaba con la información requerida.

4.5 Escenarios

Una vez explicados los distintos paneles que soporta Grafana, presentaremos los cuatro distintos escenarios en los que se va a dividir el análisis de red. Se entenderá por escenario cada cuadro de mandos con paneles y campos relacionados entre sí; todos ellos permiten analizar el estado de la red. Como se comentó, se va a seguir una organización similar a la que ofrece Packet Analyzer de Riverbed.

4.5.1 Análisis básico

Este tipo de análisis será el más sencillo de todos, ya que simplemente analizará los campos más básicos de la captura de tráfico.

Lo primero, que será común en todos los escenarios establecidos, es representar la llegada de los paquetes como un gráfico temporal que permita ver de manera clara la distribución de los paquetes en el tiempo. Esto permitirá detectar picos de tráfico o patrones con los que poder determinar de un primer vistazo el estado de la red. Los campos analizados serán los siguientes:

- Análisis temporal:
    - Llegada de paquetes ordenados por su marca de tiempo.
    - Llegada de paquetes marcados por sus direcciones MAC origen.
    - Llegada de paquetes marcados por sus direcciones MAC destino.
    - Llegada de paquetes marcados por los pares de direcciones MAC origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por su tipo Ethernet.
    - Llegada de paquetes marcados por su dirección IP origen.
    - Llegada de paquetes marcados por su dirección IP destino.
    - Llegada de paquetes marcados por su protocolo IP.
    - Llegada de paquetes marcados por los pares de direcciones IP origen y destino.
    - Llegada de paquetes marcados por su longitud.
    - Percentiles del tamaño de los paquetes.
- Análisis mediante diagrama tipo tarta:
    - Cantidad de paquetes marcados por sus direcciones MAC de origen.
    - Cantidad de paquetes marcados por sus direcciones MAC de destino.
    - Cantidad de paquetes marcados por su Ethertype.
    - Cantidad de paquetes marcados por sus pares de direcciones MAC origen y destino.
    - Cantidad de paquetes marcados por sus direcciones IP origen.
    - Cantidad de paquetes marcados por sus direcciones IP destino.
    - Cantidad de paquetes marcados por su protocolo IP.
    - Cantidad de paquetes marcados por sus pares de direcciones IP origen y destino.
- Otros diagramas:
    - Histogramas con la longitud de los paquetes.
    - Singlestat panel con valores máximo, mínimo y medio de longitud de paquetes.

4.5.2 Análisis de red de área local (LAN)

Este tipo de análisis es más específico que el anterior y se centra en el estado de una red de área local. Para ello, se hará un análisis de protocolos a nivel de direcciones MAC, del protocolo ARP, UDP, DNS e ICMP.

Los campos analizados serán los siguientes:

- Análisis temporal:
    - Llegada de paquetes ordenados por su marca de tiempo.
    - Llegada de paquetes marcados por sus direcciones MAC origen.
    - Llegada de paquetes marcados por sus direcciones MAC destino.
    - Llegada de paquetes de difusión de Ethernet.
    - Llegada de paquetes de tráfico ARP.
    - Llegada de paquetes de tráfico UDP.
- Análisis mediante diagrama tipo tarta:
    - Cantidad de paquetes marcados por sus direcciones MAC de origen.
    - Cantidad de paquetes marcados por sus direcciones MAC de destino.
    - Cantidad de paquetes marcados por su tipo de trama Ethernet.
    - Cantidad de paquetes marcados por sus pares de direcciones MAC origen y destino.
    - Cantidad de paquetes marcados por sus dirección origen IP de ARP.
    - Cantidad de paquetes marcados por su dirección destino IP de ARP.
    - Cantidad de paquetes marcados por su dirección origen MAC de ARP.
    - Cantidad de paquetes marcados por su dirección destino MAC de ARP.
    - Cantidad de paquetes marcados por sus respuestas de DNS.
    - Cantidad de paquetes marcados por su CNAME de DNS.
- Otros diagramas:
    - Tabla con los CNAME y su frecuencia de aparición.
    - Tabla con el ICMP Type y su frecuencia de aparición.
    - Diagrama de relación entre IP origen y destino.

4.5.3 Rendimiento y Errores

En este escenario se van a presentar medidas de rendimiento así como de posibles errores en las transmisiones, por ello se van a analizar campos como el tiempo de vida de los paquetes IP, las banderas y tamaño de las ventanas del protocolo TCP.

- Análisis temporal:
    - Llegada de paquetes ordenados por su marca de tiempo.
    - Llegada de paquetes marcados por sus direcciones IP origen.
    - Llegada de paquetes marcados por sus direcciones IP destino.

    - Llegada de paquetes marcados por los pares de direcciones IP origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por su tiempo de vida de IP.
    - Llegada de paquetes marcados por las banderas de TCP (FIN, Reset…).
    - Llegada de paquetes marcados por el tamaño de la ventana de TCP.
- Análisis diagrama tipo tarta:
    - Cantidad de paquetes marcados por sus direcciones IP origen.
    - Cantidad de paquetes marcados por sus direcciones IP destino.
    - Cantidad de paquetes marcados por su protocolo IP.
    - Cantidad de paquetes marcados por sus pares de direcciones IP origen y destino.
    - Cantidad de paquetes marcados por su tiempo de vida de IP.
- Otros diagramas:
    - Estadísticas (singlestat) de TCP (Segmentos perdidos, fuera de orden, etc.).

4.5.4 Interlocutores y conversaciones

El objetivo de este escenario es el de visualizar qué protocolos y máquinas están generando un mayor tráfico. Aunque en el resto de escenarios también se hace un breve estudio de qué máquinas se están comunicando entre sí para poner en contexto el análisis, este escenario se centra aún más en ese propósito. Los campos analizados serán los siguientes:

- Análisis temporal:
    - Llegada de paquetes ordenados por su marca de tiempo.
    - Llegada de paquetes marcados por sus direcciones MAC origen.
    - Llegada de paquetes marcados por sus direcciones MAC destino.
    - Llegada de paquetes marcados por los pares de direcciones MAC origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por su tipo Ethernet.
    - Llegada de paquetes marcados por sus direcciones IP origen.
    - Llegada de paquetes marcados por sus direcciones IP destino.
    - Llegada de paquetes marcados por sus pares de direcciones IP origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por su puerto de origen TCP.
    - Llegada de paquetes marcados por su puerto de destino TCP.
    - Llegada de paquetes marcados por sus pares TCP origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por su puerto de origen UDP.
    - Llegada de paquetes marcados por su puerto de destino UDP.

    - Llegada de paquetes marcados por sus pares UDP origen y destino en comunicación unilateral.
    - Llegada de paquetes marcados por sus direcciones IPv4 de DNS.
- Análisis diagramas de tipo tarta:
    - Cantidad de paquetes marcados por sus direcciones MAC origen.
    - Cantidad de paquetes marcados por sus direcciones MAC destino.
    - Cantidad de paquetes marcados por su tipo de trama Ethernet.
    - Cantidad de paquetes marcados por sus pares de direcciones MAC origen y destino.
    - Cantidad de paquetes marcados por sus direcciones IP origen.
    - Cantidad de paquetes marcados por sus direcciones IP destino.
    - Cantidad de paquetes marcados por su protocolo IP.
    - Cantidad de paquetes marcados por sus pares de direcciones IP origen y destino.
    - Cantidad de paquetes marcados por su puerto de origen TCP.
    - Cantidad de paquetes marcados por su puerto de destino TCP.
    - Cantidad de paquetes marcados por sus pares TCP origen y destino.
    - Cantidad de paquetes marcados por su puerto de origen UDP.
    - Cantidad de paquetes marcados por su puerto de destino UDP.
    - Cantidad de paquetes marcados por sus pares UDP origen y destino.
- Otros diagramas:
    - Tabla con CNAME de DNS.

4.6 Conclusiones

En este capítulo se han sentado las bases del proyecto realizado. Se ha estudiado los motivos principales del uso de ficheros en formato CSV en lugar de JSON; seleccionando los ficheros CSV por ocupar un menor tamaño, así como un mayor tiempo de procesamiento al obtener un fichero en formato JSON con Tshark.

Además de esto, se han presentado los campos que serán utilizados para realizar el análisis de red con su posterior representación utilizando Grafana. En esta representación, se han establecido cuatro escenarios distintos que permiten analizar los paquetes transmitidos por la red de manera organizada y clara.

5 Integración, pruebas y resultados

El objetivo de este capítulo es el de implementar el sistema que permita analizar el tráfico de red. Para ello, el capítulo se dividirá en dos partes claramente diferenciadas: por una parte, el análisis de trazas PCAP y, por otra parte, análisis de tráfico en tiempo real.

Además de esto, en las dos situaciones planteadas, se va a distinguir entre un sistema propio desarrollado con las herramientas comentadas previamente y otro sistema profesional con un equipo de desarrollo tras él.

5.1 Análisis de trazas

5.1.1 Moloch

Como ya se introdujo en el estado del arte del capítulo 2, Moloch es una herramienta que permite el análisis de ficheros PCAP y que guarda una importante similitud con el sistema planteado en los capítulos 3 y 4 ya que utiliza Elasticsearch como motor de búsqueda de su sistema. Moloch es un proyecto open source que permite capturar tráfico e indexarlo, pero en lugar de utilizar Grafana, han desarrollado su propia interfaz web donde visualizar los datos.

Por lo comentado, para implementar este sistema, es necesario primero tener un nodo Elasticsearch corriendo donde se almacenarán los datos. Más tarde, se tiene que instalar Moloch. Una ventaja importante, es la de poder cargar ficheros PCAP que se tengan almacenados. Al utilizar Molcoh, no es necesario pasar la captura a un fichero CSV para indexarlo de esta manera en Elasticsearch, sino que, directamente, se puede trabajar con ella en formato PCAP.

Para acceder a esta visualización, es suficiente con acceder a la dirección del equipo local (*localhost*) predeterminada. Por defecto, Moloch presenta varios campos y permite añadir aquellos que sean necesarios para analizar el tráfico. Como se puede ver en la Figura 5-1, en la que se puede ver el ejemplo de análisis de una captura de paquetes realizada utilizando Tshark. Esta captura se almacena en disco en formato PCAP, sin necesidad de convertirlo a otro tipo de fichero JSON o CSV. Posteriormente, se puede utilizar un comando en una consola Linux para cargar el fichero PCAP directamente en Moloch. En el diseño de la aplicación web en la que se analiza la captura aparece en formato texto la información de los distintos protocolos, una sencilla gráfica en la que se muestra la llegada temporal de los paquetes, acompañado de un mapa del mundo donde, con un color más oscuro, se presenta la geolocalización de los paquetes. Además de los anteriores, Moloch ofrece la posibilidad de añadir otros campos que sean necesarios para analizar el tráfico.

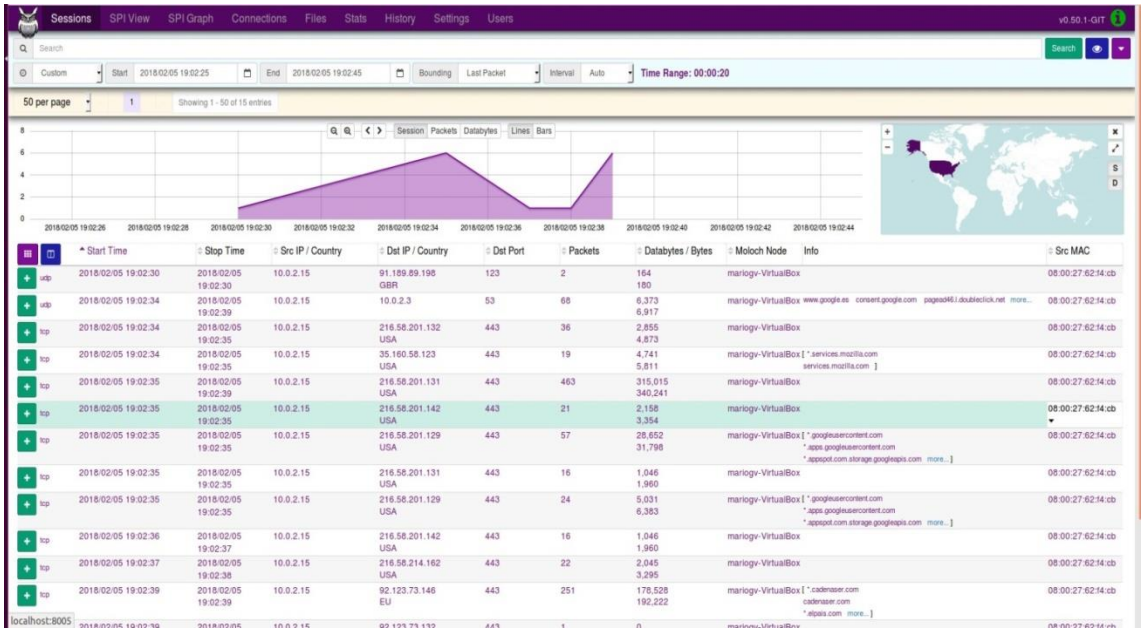

Figura 5-1 – Análisis de tráfico en Moloch

Así también, Moloch permite realizar una visualización por sesión, por paquetes o por bytes de datos recibidos. Por último, Moloch ofrece también una visualización de grafos que relacionan por ejemplo las direcciones IP o MAC que se comunican entre sí como se puede ver en la Figura 5-2:

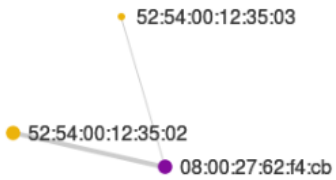

Figura 5-2 – Grafo de direcciones MAC en Moloch

Los principales problemas y limitaciones que presenta Moloch es que su funcionalidad se ve limitada por el cuadro de mandos que ofrece. Si bien es cierto que hay una cantidad suficiente de información y paneles, para conseguir un análisis más fino y exhaustivo de la red puede no resultar todo lo completo que se necesita. Además, al recibir una captura de datos completa, hay campos que pueden no ser de interés para el análisis y su tratamiento resulta innecesario.

5.1.2 Sistema basado en Elasticsearch, Logstash y Grafana

En este sistema, se tomarán capturas de tráfico previamente almacenadas y se procederá a su análisis. Se puede ver un resumen del funcionamiento del sistema en la Figura 5-3, donde se plantea ejecutar Tshark para tratar la captura de tráfico y, posteriormente, utilizar Logstash que se encargará de procesar los datos resultantes de Tshark y cargarlos en Elasticsearch, desde donde Grafana tomará la información para representarlo en un cuadro de mandos.

Figura 5-3 – Estructura del sistema

Para ello, y como se ha comentado en secciones anteriores, resulta necesario filtrar los datos de la captura para analizar solamente los campos que resultan de interés. Esto se puede realizar utilizando Tshark, recogiendo sólo algunos de sus campos en un nuevo fichero en formato CSV para, posteriormente, ser cargados en un nodo Elasticsearch que será donde se almacenen todos los datos.

Para realizar esto, se puede ejecutar un comando similar al presentado en el apartado 4.2 de esta memoria, en el que se obtenían los campos de la captura de tráfico que resultaban de interés en formato CSV.

En este punto, se utilizará Logstash, parte de la pila ELK, para cargar los datos en Elasticsearch. Para realizar esto, es necesario crear un fichero de configuración de Logstash en el que se especificarán varios parámetros necesarios para leer el fichero CSV, tratar los datos y mandarlos al nodo Elasticsearch.

El fichero de configuración se divide en tres partes fundamentales:

- **Input:** donde se le especifica el directorio en el que se encuentra el fichero del cual se quieren leer los datos y por dónde empezar a leer el fichero.
- **Filter:** en esta parte se especifica cómo hay que tratar el fichero CSV. Así, se le indicará el tipo de separador de los campos, por ejemplo un tabulador, así como las columnas del fichero, las cuales se corresponderán con los distintos campos. En esta parte, también se puede especificar algún tratamiento concreto de los datos, como por ejemplo, cambiar el tipo de los mismos, o convertir la longitud de los paquetes a un valor entero, con el objetivo de poder representarlo en Grafana.
- **Output:** por último, se le indica qué debe hacer con los datos que ha leído y procesado. En nuestro caso, queremos que sean indexados en Elasticsearch, para ello, se le indicará la dirección donde está corriendo Elasticsearch, así como un nombre para el índice.

Para cada uno de los escenarios planteados en el capítulo 4, será necesario realizar un fichero de configuración distinto que permita indexar los datos que se quieran. Se puede ver un ejemplo de fichero de configuración en el anexo C: Fichero de configuración de Logstash.

El último paso se corresponde con la representación de los datos en Grafana empleando los gráficos que se plantearon en el capítulo 4.

Para crear un nuevo cuadro de mandos en Grafana será suficiente con crear un "*data source"* (ver anexo D: Creación de un data source en Grafana), ir a la pestaña "Dashboards", seleccionar la opción "New Dashboard" y en ese momento, se podrá elegir el tipo de panel que queramos. Grafana ofrece una amplia variedad de paneles, pero es posible añadir más mediante la instalación manual de los mismos. A continuación, se presenta cómo quedaría la visualización para el caso de uno de los escenarios, por ejemplo, para el caso del escenario "Talkers & Conversations", presentado en la sección 4.5.4.

Lo primero será representar la llegada de los paquetes, los cuales irán ordenados gracias a su marca de tiempo (correspondiente con el campo frame_time_epoch de la captura de Tshark). El gráfico se puede ver en la Figura 5-4.

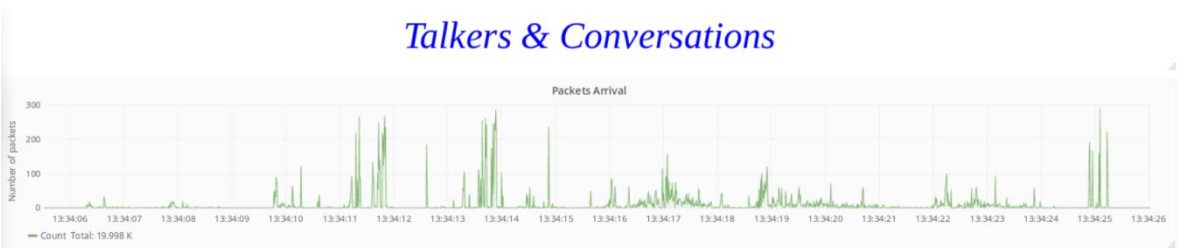

Figura 5-4 – Llegada de paquetes

Posteriormente, en el mismo cuadro de mandos, se pasa a realizar un análisis a nivel Ethernet, analizando algunos campos como las direcciones MAC de origen y destino, así como el Ethertype. En la Figura 5-5 se puede ver el bloque que representa el análisis de protocolo Ethernet, donde, por una parte encontramos la llegada de los paquetes con determinadas direcciones MAC de origen y de destino, así como su Ethertype. A esto le sigue una representación utilizando diagramas de tipo tarta.

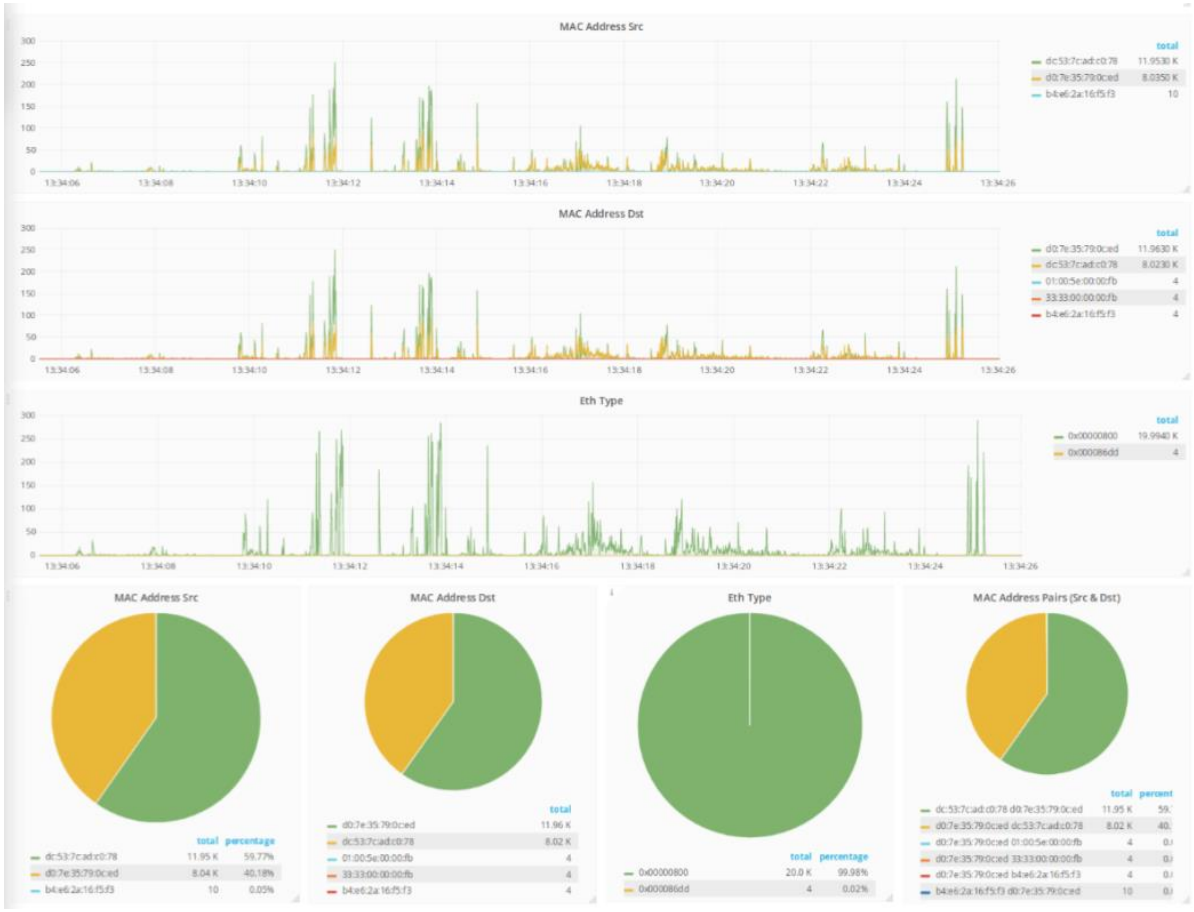

Figura 5-5 – Análisis MAC

De manera similar, a nivel de IP, en este caso se analiza también el protocolo IP de los paquetes como se presenta en la Figura 5-6.

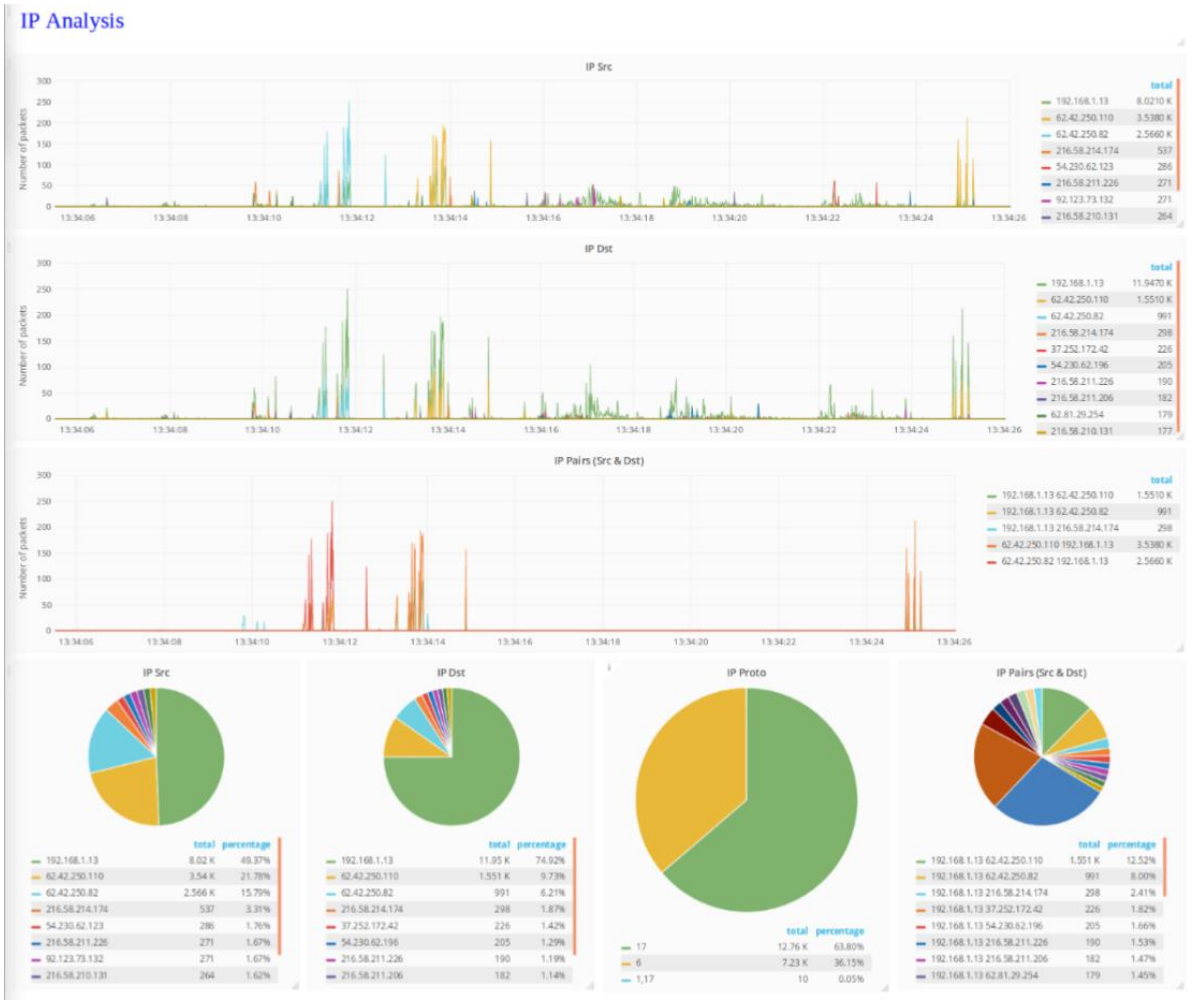

Figura 5-6 – Análisis IP

En el caso de otros escenarios, podemos ver algunas representaciones que resultan de interés, como por ejemplo, un diagrama de conversaciones MAC del escenario "LAN Analysis" representado en la Figura 5-7:

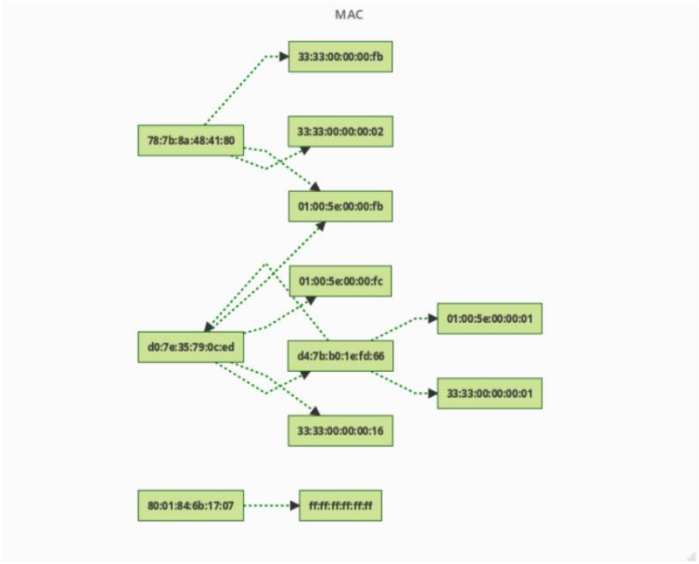

Figura 5-7 – Diagrama de direcciones MAC

Por último, en la Figura 5-8, podríamos ver unas estadísticas resultantes del estudio de TCP. Con esto, podemos representar algunos campos como la cantidad de banderas en paquetes TCP en la captura y cuáles de ellos son de tipo pérdida de segmento, fuera de orden, etc.

Figura 5-8 – Estadísticas TCP

Con lo planteado anteriormente ya tendríamos un sistema propio capaz de analizar capturas previamente almacenadas haciendo uso de Tshark para filtrar la captura, Logstash para procesar la información, Elasticsearch para almacenar los datos y, por último, Grafana para representar la información.

5.1.3 Comparativa

A continuación, se presenta en la Tabla 5-1 una comparativa entre los dos sistemas planteados:

Tabla 5-1: Comparativa sistemas de análisis de trazas

	Moloch	**ES + Logstash + Grafana**
Escalabilidad	Alta, al estar basado en ES.	Alta, al estar basado en ES.
Implementación	Sencilla y el proceso de instalación es largo. Es fácil cargar nuevos datos.	Requiere definir los ficheros de configuración de Logstash y tratar las capturas con Tshark.
Funcionalidad	Limitado por las funciones ya implementadas.	Extensible, puesto que tenemos control total sobre el sistema.
Visualización	Pocos gráficos; los mínimos para realizar un análisis de la red.	Muy flexible gracias al uso de Grafana.

5.2 Análisis en tiempo real

5.2.1 Sistema basado en Elasticsearch, Packetbeat y Grafana

En este caso, se va a utilizar una funcionalidad aportada por el equipo de Elastic. Se trata de Packetbeat el cual es un analizador del tráfico de red en tiempo real.

Para poner en marcha Packetbeat es suficiente con descargarlo y ejecutarlo como servicio, de esta forma, se quedará corriendo mientras que se van cargando todos los datos recolectados en un índice Elasticsearch que genera automáticamente.

Evidentemente, Elastic plantea este servicio como parte de la pila ELK, para terminar visualizando los datos en Kibana, sin embargo, por las razones explicadas en el capítulo 3, se seguirá utilizando Grafana para la visualización y creación de los cuadros de mandos.

Con esto, tenemos un sistema de análisis de tráfico en tiempo real implementado con la misma idea de tomar a Elasticsearch como motor de indexado del sistema.

5.2.2 Sistema basado en Tshark, Elasticsearch y Grafana

Se va a desarrollar un sistema capaz de ir capturando tráfico en tiempo real y proceder a su análisis intentando que no exista una demora demasiado importante entre la captura de los datos y su visualización.

Para ello, se ha realizado un script en Bash que permita la automatización de estas tareas. Este script debe realizar las siguientes tareas:

- Crear un índice de ES y asignarle el mapeo deseado.
- Capturar tráfico utilizando Tshark.
- Cargar el fichero con la captura de los paquetes en ES.
- Reindexar los datos al índice que tiene el mapeo correcto.

El mapeo del índice de ES consiste en definir cómo va a ser el documento que se va a indexar, especificando los campos que va a tener y el tipo de los datos del documento. Al cargar los datos, Elasticsearch le asigna automáticamente un mapeo por defecto, en el cuál los campos se definen de tipo texto. Esto supone un problema ya que, cuando se quieren visualizar los datos en Grafana, si se quiere representar, por ejemplo, la longitud de los paquetes de la captura, nos va a dar error, ya que estamos esperando ver un tipo entero cuando lo que tenemos es un texto. Por esto, resulta necesario modificar el mapeo de campos y crear uno que nos permita tener en cuenta cada tipo de dato que vamos a visualizar, para poder tratarlo de manera adecuada.

Hay que tener en cuenta también la cantidad de tráfico que está generando la máquina donde se ejecuta Tshark. Así, se podrá hacer una estimación de cuántos paquetes cargar en Elasticsearch cada vez que se llama a la API de ES para indexar los datos en el índice. De esta forma, una máquina que genere más tráfico, deberá aumentar la cantidad de paquetes, en cambio, si genera poco tráfico, habría que disminuir la cantidad, pues retrasaría mucho la visualización.

En el caso del script desarrollado (ver anexo E: Script para automatizar el proceso de captura e indexación), se ha decidido que cada 1.000 paquetes capturados, se escriban los datos en un fichero (en este caso en formato JSON) y se pase a llamar a la API utilizando para ello cURL. De esta forma, usando el método PUT, se cargan los datos en el índice. En este caso, ya no es necesario utilizar Logstash, pues vamos a ser nosotros los que nos encarguemos, utilizando cURL de cargar los datos en ES y de definir el mapeo que necesitemos.

Una vez que se está ejecutando el script, se puede ir directamente a Grafana donde, tras configurar el origen de datos, es posible ir visualizando los datos tras su llegada. Hay que tener en cuenta que, en la configuración del cuadro de mandos, es necesario indicar que se refresque periódicamente, para que los paquetes que se siguen capturando en segundo plano puedan ir siendo representados.

A esto, se le puede aplicar, evidentemente, los escenarios que fueron planteados en capítulos anteriores, pues el objetivo sigue siendo común, salvo que, ahora, las visualizaciones son dinámicas al ir visualizando la llegada de los nuevos paquetes.

5.2.3 Comparativa

Con los dos sistemas vistos se presenta en la Tabla 5-2 una comparativa entre ambos sistemas:

Tabla 5-2: Comparativa sistemas de análisis en tiempo real

	ES + Packetbeat + Grafana	**Tshark + ES + Grafana**
Escalabilidad	Alta, al estar basado en ES.	Alta, al estar basado en ES.
Implementación	Muy sencilla. Descargar y ejecutar (teniendo en cuenta sólo Packetbeat).	Dificultad media al tener que desarrollar el script en Bash y estudiar el mapeo más adecuado para el sistema.
Funcionalidad	Ofrece una gran cantidad de campos para su análisis, aunque quizá sean demasiados, ya que muchos pueden no resultar de interés.	Alta, al poder configurar y utilizar cada uno de los componentes del sistema de manera individual, explotando las características de cada uno.
Visualización	Alta calidad gracias al uso de Grafana.	Alta calidad gracias al uso de Grafana.

5.3 Conclusiones

En este capítulo se han estudiado un total de cuatro implementaciones distintas, dos de ellas para el análisis de trazas y otras dos para el análisis en tiempo real, pero todas basadas en el uso de Elasticsearch como motor de indexado.

Se ha visto cómo Grafana supone un elemento clave, pues da mayor flexibilidad y una mayor riqueza de gráficos. Esto, unido a una implementación relativamente sencilla, hacen del sistema basado en Elasticsearch y Grafana, el elegido para realizar el sistema que permita realizar un análisis de red.

Por otra parte, en lo que se refiere al análisis en tiempo real, los dos sistemas presentados ofrecen un buen análisis, sin embargo, si se quiere obtener un mayor control sobre los campos que se desean analizar también habría que recurrir al sistema desarrollado con Tshark, Elasticsearch y Grafana.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este Trabajo Fin de Grado, se ha implementado un sistema capaz de analizar grandes cantidades de tráfico de una manera rápida, sencilla y muy visual, lo cual era el objetivo principal del proyecto. Con esto, se ha conseguido desarrollar un sistema alejado del concepto tradicional de análisis de red, pero empleando herramientas de-facto como Tshark/Wireshark para la captura del tráfico.

Se ha demostrado como Elasticsearch es un motor de indexación de datos que se adecua perfectamente a los objetivos de desarrollo de este trabajo, siendo pieza clave en todas las opciones propuestas durante la realización del proyecto. Por la sencillez de uso, fácil implementación y compatibilidad con las ideas planteadas, Elasticsearch se postula como la mejor opción para realizar el sistema planteado.

Con todo lo anterior, se han planteado cuatro sistemas distintos todos ellos con el fin de analizar tráfico de red: dos de ellos para análisis de trazas previamente capturadas y almacenadas, y otros dos como sistemas de análisis de red en tiempo real. Para cada tipo de sistema planteado, se han comparado soluciones disponibles actualmente con sistemas de realización propia en los que se pueda visualizar exactamente la información necesaria para el análisis de redes.

La diferencia entre los sistemas planteados ha residido precisamente en que sistemas como Moloch ofrecen un servicio de calidad, pero limitados en cuanto a presentación visual de los datos pues, en ocasiones, resultan demasiado sencillos. Además, no ofrece posibilidad de ampliar su funcionalidad en ningún momento. Por el contrario, un sistema propio desarrollado sobre Elasticsearch y Grafana, ha permitido obtener un sistema final más versátil, personalizable y visual.

Para el desarrollo de este Trabajo Fin de Grado habría que tener en cuenta el papel que han jugado asignaturas vistas a lo largo de la carrera que han permitido desarrollar el proyecto planteado. Por ejemplo, asignaturas como Arquitectura de Redes I, donde se establecieron los conceptos fundamentales de las redes de computadores, así como el estudio de los protocolos que han sido utilizados para definir los distintos escenarios planteados en la sección 4.5. Por supuesto, de igual forma habría que destacar Arquitectura de Redes II, donde se estudió el nivel de enlace que también ha sido visualizado en este trabajo así como un análisis de redes LAN.

Además de esto, habría que destacar asignaturas como Redes Multimedia donde se estudiaron protocolos como TCP y UDP y se hicieron medidas de calidad de servicio, además de servir como introducción a Tshark, y permitir profundizar en el aprendizaje y uso de Wireshark.

Finalmente, cabe destacar la asignatura de Sistemas Distribuidos, donde se realizó un estudio de aplicaciones web vistas también en el desarrollo de este TFG. Además, en esta asignatura se estudiaron varios lenguajes de programación y de consultas a bases de datos entre los que cabe destacar SQL, útil para realizar consultas a InfluxDB.

6.2 Trabajo futuro

El desarrollo realizado en este Trabajo Fin de Grado ha servido como punto de partida para definir un sistema de análisis de red utilizando herramientas de tratamiento masivo de

datos como son Elasticsearch y Grafana. Es por ello que da lugar a la posibilidad de mejorar el sistema por varios caminos distintos entre los que cabe destacar:

- El rendimiento del sistema, estudiando detenidamente el proceso de indexación de los datos en los nodos de Elasticsearch y presentando una solución que permita aumentar la tasa de datos que se insertan en el índice de ES. Con esto se conseguiría mejorar el sistema adecuándolo más a situaciones en las que existen restricciones temporales a la hora de realizar el análisis de red. Se podrían ampliar las funcionalidades del sistema, añadiendo mayor cantidad de representaciones o, también, planteando nuevos escenarios de análisis que se pudieran especializar en situaciones más concretas si fueran necesarias.
- La resolución de problemas, como el problema de generar dinámicamente diagramas de conversación entre direcciones IP o direcciones MAC de la red.

Por todo ello, este trabajo final de grado puede ser el punto de partida para futuros trabajos final de grado de otros estudiantes que finalicen tanto el Grado en Ingeniería Informática como el Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación.

Referencias

[1] Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. [On-line] https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

[2] Wireshark Docs. A brief history. [On-line] https://www.wireshark.org/docs/wsug_html_chunked/ChIntroHistory.html

[3] Riverbed SteelCentral Packet Analyzer [on-line] https://www.riverbed.com/gb/products/steelcentral/steelcentral-packet-analyzer.html

[4] Moloch Site. [On-line] https://molo.ch/

[5] Arquitectura de Packetbeat. [On-line] https://www.elastic.co/products/beats/packetbeat

[6] GNU Bash. [On-line] https://gnu.org/software/bash/

[7] Elastic Web Page. [On-line] https://www.elastic.co/

[8] The future of Compass & Elasticsearch. Shay Banon. 2013. [On-line] https://web.archive.org/web/20130827121405/http://www.kimchy.org/the_future_of_compass/

[9] Apache Lucene. [On-line] https://lucene.apache.org/core/7_3_0/index.html

[10] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

[11] Elasticsearch Past Releases. Elasticsearch Oficial Web Page. [On-line] https://www.elastic.co/downloads/past-releases

[12] JSON. [On-line] https://www.json.org/

[13] cURL. [On-line] https://curl.haxx.se/

[14] InfluxData Web Page. [On-line] https://www.influxdata.com/time-series-platform/

[15] Go. [On-line] https://golang.org/

[16] Csv-to-influxdb script. [On-line] https://github.com/jpillora/csv-to-influxdb

[17] Graphite Web Page. [On-line] https://graphiteapp.org/

[18] Grafana Web Page. [On-line] https://grafana.com/

[19] Wireshark Docs. Display Filter Reference. [On-line] https://www.wireshark.org/docs/dfref/

[20] RFC 1034 – Domain Names. 1987. [On-line] https://tools.ietf.org/html/rfc1034/

[21] Mermaid Syntax. [On-line] https://mermaidjs.github.io/

Anexos

A Modificación de la función csv-to-influxdb

Debido a que el módulo *csv-to-influxdb* sólo estaba preparado para recibir las marcas de tiempo de los paquetes en formato cadena de texto y Tshark ofrece esta información en formato valor de coma flotante, es necesario modificar el módulo añadiendo las líneas de código definidas en la Tabla A-1.

Como modificaciones, se ha añadido un nuevo bloque if de tal forma que si recibe un string y lo que se codifica en este string es de tipo float, se divide en dos la cadena, de tal forma que lo de antes del punto se corresponderán con los segundos y lo de después serán los nanosegundos (líneas 230 a 240).

Con esto, se genera una nueva variable t en formato tiempo Unix y se establece como columna que será utilizada como marca de tiempo (líneas (241 a 248).

De esta forma, si el formato recibido es de tipo coma flotante, se puede componer una nueva marca de tiempo para los paquetes en formato cadena de texto.

Tabla A-1 – Modificaciones sobre la función csv-to-influxdb

```
230         if !conf.ForceString && floatRe.MatchString(r) {
231                 // Unix-timestamp parsing
232                 s := strings.Split(r, ".")
233                 sec, err := strconv.ParseInt(s[0], 10, 64)
234                 if err != nil {
235                         panic(err)
236                 }
237                 nanosec, err := strconv.ParseInt(s[1], 10, 64)
238                 if err != nil {
239                         panic(err)
240                 }
241                 t := time.Unix(sec, nanosec)
242                 if conf.TimestampColumn == h {
243                         ts = t //the timestamp column!
244                         continue
245                 }
246                 fields[h] = t
247         }
248 }
```

Para ejecutar este módulo será necesario ejecutar el comando presentado en la Tabla A-2:

Tabla A-2 – Ejecutar csv-to-influxdb

```
go run ./main.go –d test –ts frame_time_epoch –tf “[0-9]*(\.[0-9]+)?” prueba.csv
```

B Generación de la sintaxis Mermaid

Para generar la sintaxis Mermaid se ha desarrollado un código en Python que procese la información del fichero CSV y se obtenga una sintaxis Mermaid válida. De esta manera, se le introduce el nombre del fichero con la ruta y se le indica en qué columnas del fichero están las direcciones MAC e IP que se quieren tratar (líneas 4 a 13). Posteriormente, se lee el fichero y se va metiendo en cuatro listas las direcciones. Dos de ellas para direcciones MAC origen y destino y otras dos para direcciones IP origen y destino (líneas 19 a 25). Después se hace un '*set*' para eliminar los pares duplicados y se pinta añadiendo una flecha entre las dos direcciones IP, obteniendo así la sintaxis válida para Mermaid (líneas 37 a 44). De igual manera con las direcciones MAC (líneas 50 a 57). Esto se puede ver en la Tabla B-3:

Tabla B-3 – Script en Python para generar la sintaxis Mermaid

```
1  #!/usr/bin/env python
2  # Python script to read a CSV file and generate marmaid syntax
3  import csv, operator
4  FILE_TO_READ = '/home/mario/Escritorio/Capturas y config
5  files/lananalysis.csv' # Name of valid CSV to read
6  ip_src = []            # Empty list for source IP
7  ip_dst = []            # Empty list for destination IP
8  eth_src = []           # Empty list for source MACs
9  eth_dst = []           # Empty list for destination MACs
10 ip_src_row = 1         # Number of source IP column
11 ip_dst_row = 2         # Number of destination IP column
12 eth_src_row = 4        # Number of ethernet source column
13 eth_dst_row = 5        # Number of ethernet destination column
14
15 ip_src_res = []
16 ip_dst_res = []
17 j=0
18 # Read CSV file
19 with open(FILE_TO_READ) as CSVFile:
20     read = csv.reader(CSVFile, delimiter='\t')
21     for row in read:
22         ip_src.append(row[ip_src_row])
23         ip_dst.append(row[ip_dst_row])
24         eth_src.append(row[eth_src_row])
25         eth_dst.append(row[eth_dst_row])
26
27     # Delete header
28
29     #del ip_src[0]
30     #del ip_dst[0]
31     #del eth_src[0]
32     #del eth_dst[0]
33
34     print("IP Address")
35     print("-----------------------------------")
36
37     s = set([])
38     for i in range(len(ip_src)):
39         if(ip_src[i] in ["", None]): continue
40         if(ip_dst[i] in ["", None]): continue
41         a = ip_src[i] + "-.->" + ip_dst[i]
42         if (a in s): continue
```

```
43          print(a)
44          s.add(a)
45
46      print("")
47      print("MAC Address")
48      print("-------------------------------------")
49
50      s_mac = set([])
51      for i in range(len(eth_src)):
52          if(eth_src[i] in ["", None]): continue
53          if(eth_dst[i] in ["", None]): continue
54          b = eth_src[i] + "-.->" + eth_dst[i]
55          if (b in s_mac): continue
56          print(b)
57          s_mac.add(b)
```

C Fichero de configuración de Logstash

En el fichero de configuración aparecen tres partes claramente diferenciadas. Por una parte la entrada (input) donde se elige la ruta del fichero CSV a tratar (línea 4). Se le indica que comience a leer el fichero por el principio (línea 5).

En la etapa de filtrado (filter) se le indica cómo tratar el CSV, indicándole el tipo de separador, en este caso, un tabulador y el nombre de las columnas.

En el mutate se le indica si se quiere convertir algún campo, en este caso, pasamos la longitud de los paquetes a un entero (línea 28).

En la etapa de fecha (date), se convierte el campo frame_time_epoc a tiempo UNIX y se pasa a un nuevo campo denominado frame_time_epoch_packets (líneas 31 y 32).

Por último, en el bloque de salida (output) se le va a indicar dónde almacenar los datos tratados. Se le indica que se quieren indexar, la dirección donde se encuentra Elasticsearch y el nombre del índice. También se le puede indicar el tipo de documento, para poder clasificarlos y los trabajadores (líneas 36 a 43). Se puede ver en la Tabla C-4:

Tabla C-4 – Fichero de configuración de Logstash

```
1   input {
2         file {
3               path => ["/home/mario/Escritorio/basicanalysis.csv"]
4               start_position => "beginning"
5               sincedb_path => "/dev/null"
6         }
7   }
8   filter {
9         csv {
10              separator => "      "
11              columns => [
12                    "frame_time_epoch",
13                    "ip.src",
14                    "ip.dst",
15                    "ip.proto",
16                    "eth.src",
17                    "eth.dst",
18                    "eth.type",
19                    "tcp.srcport",
20                    "tcp.dstport",
21                    "udp.srcport",
22                    "udp.dstport",
23                    "frame.len"
24              ]
25        }
26
27        mutate {
28      convert => { "frame.len" => "integer" }
29        }
30        date {
31              match => ["frame_time_epoch", "UNIX"]
32              target => "frame_time_epoch_packets"
33        }
34
35  }
36  output {
37        stdout { codec => rubydebug }
```

```
38          elasticsearch {
39                  action => "index"
40                  hosts => "localhost:9200"
41                  index => "basic_analysis"
42                  document_type => "network_analysis"
43                  workers => 1
44          }
45  }
46
```

D Creación de un data source en Grafana

Una vez descargado y ejecutado Grafana, es necesario ir a la pestaña "Data Sources" para configurar el origen de los datos para que Grafana tome la información desde ahí para su posterior representación. Será necesario indicarle el nombre deseado para ese origen de datos, el tipo de base de datos que es, en este caso es de tipo Elasticsearch, así como la dirección URL donde ES atiende a peticiones. Por último, habrá que indicar el nombre del índice de ES donde se encuentran los datos que se desean representar, así como el nombre del campo que actuará como marca de tiempo. Se puede ver en la Figura D-1:

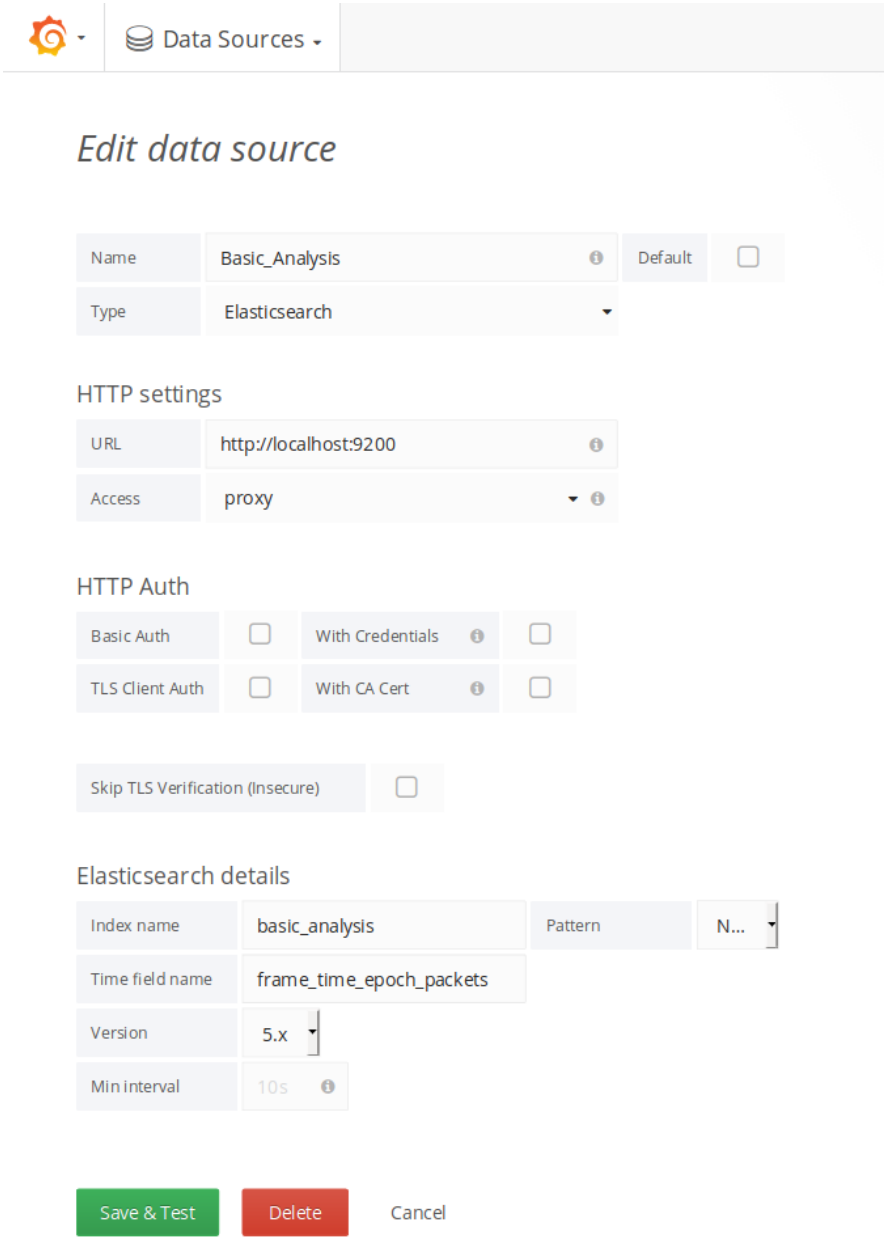

Figura D-1 – Creación de una fuente de datos

E Script para automatizar el proceso de captura e indexación

Con el objetivo de tener un sistema capaz de capturar tráfico de internet utilizando Tshark e ir indexando los paquetes en un nodo Elasticsearch, se desarrolló un script que automatizara dicho proceso. Lo primero que se hace es crear el fichero donde queremos que se almacene la captura y establecemos dos variables en las que se va a indicar el nombre de la interfaz donde se quiere capturar y el número de paquetes por captura (líneas 4 a 7). Después se genera el mapeo que resulte apropiado para nuestro trabajo (líneas 10 a 118). Posteriormente, en un bucle while, se invoca a Tshark para que realice la captura y diseccione para quedarnos sólo con los campos que resultan de interés (líneas 120 a 126). Por último, se carga el fichero con los paquetes con los campos de interés y se reindexa desde el índice creado automáticamente por Elasticsearch al que previamente hemos creado con el mapeo adecuado (líneas 129 a 145). Se puede ver en la Tabla E-5:

Tabla E-5 – Script de automatización del proceso de captura e indexación

```
1   #!/bin/bash
2   # Script que realiza la funcion principal del proyecto
3   # Creamos el fichero sobre el que vamos a escribir la captura
4   sudo touch captura.pcap
5
6   INTERFACE="wlan0"
7   PACKETS_NUMBER="1000"
8
9   # Cambiamos el mapping
10  curl -XPUT 'http://localhost:9200/tfg2?pretty' -H 'Content-Type:
11  application/json' -d'
12  {
13    "mappings": {
14      "pcap_file": {
15        "properties": {
16          "layers": {
17            "properties": {
18              "eth_dst": {
19                "type": "keyword",
20                "fields": { "keyword": { "type": "keyword" } }
21              },
22              "eth_src": {
23                "type": "keyword",
24                "fields": {
25                  "keyword": {
26                    "type": "keyword"
27                  }
28                }
29              },
30              "frame_time_epoch": {
31                "type": "date",
32                "format": "epoch_millis",
33                "fields": {
34                  "keyword": {
35                    "type": "date"
36                  }
37                }
38              },
39              "frame_len": {
40                  "type": "integer",
```

```
                "fields": {
                  "keyword": {
                    "type": "integer"
                  }
                }
              },
              "ip_dst": {
                "type": "ip",
                "fields": {
                  "keyword": {
                    "type": "ip"
                  }
                }
              },
              "ip_proto": {
                  "type": "text",
                  "fields": {
                    "keyword": {
                      "type": "keyword"
                    }
                 }
              },
              "ip_src": {
                "type": "ip",
                "fields": {
                  "keyword": {
                    "type": "ip"
                  }
                }
              },
              "tcp_dstport": {
                "type": "integer",
                "fields": {
                  "keyword": {
                    "type": "integer"
                  }
                }
              },
              "tcp_srcport": {
                "type": "integer",
                "fields": {
                  "keyword": {
                    "type": "integer"
                  }
                }
              },
              "udp_dstport": {
                "type": "integer",
                "fields": {
                  "keyword": {
                    "type": "integer"
                  }
                }
              },
              "udp_srcport": {
                "type": "integer",
                "fields": {
                  "keyword": {
                    "type": "integer"
```

```
                }
              }
            }
          }
        },
        "timestamp": {
          "type": "date",
          "format": "epoch_millis",
          "fields": {
            "keyword": {
              "type": "date"
            }
          }
        }
      }
    }
  }
}
'
while [ 1 ]; do
      sudo tshark -i $INTERFACE -c $PACKETS_NUMBER -w captura.pcap

# Capturamos 1000 paquetes en el fichero creado
      tshark -e "frame.time_epoch" -e "ip.src" -e "ip.dst" -e
"ip.proto" -e "frame.len" -e "eth.src" -e "eth.dst" -e "tcp.srcport" -e
"tcp.dstport" -e "udp.srcport" -e "udp.dstport" -T ek -r captura.pcap >
captura.json

# Subimos el fichero a ElasticSearch
  curl -XPUT 'http://localhost:9200/_bulk?pretty' -H 'Content-Type:
application/json' --data-binary @captura.json
# Reindexamos desde el indice por defecto al que se sube la informacion
hacia el nuevo indice con el mapping correcto
      curl -XPOST 'localhost:9200/_reindex?pretty' -H 'Content-Type:
application/json' -d'
            {
              "source": {
                "index": "packets-2018-02-13"
              },
              "dest": {
                "index": "tfg2",
                "version_type": "internal"
              }
            }
            '
done


```