TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Implementación en tiempo real de sistemas de identificación de tráfico de red

Autor
Álvaro Maximino Linares Herrera

Directores
Jesús Esteban Díaz Verdejo

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 11 de septiembre de 2017

Implementación en tiempo real de sistemas de identificación de tráfico de red

Autor
Álvaro Maximino Linares Herrera

Directores
Jesús Esteban Díaz Verdejo

Implementación en tiempo real de sistemas de identificación de tráfico de red

Álvaro Maximino Linares Herrera

Palabras clave: inspección profunda de paquetes, bro, flujos, identificación, clasificación, red, tráfico

Resumen

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Álvaro Maximino Linares Herrera**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76669401M, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Álvaro Maximino Linares Herrera

Granada a 11 de septiembre de 2017.

D. **Jesús Esteban Díaz Verdejo**, Profesor del Departamento Teoría de la Señal, Telemática y Comunicaciones (TSTC) de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Implementación en tiempo real de sistemas de identificación de tráfico de red***, ha sido realizado bajo su supervisión por **Álvaro Maximino Linares Herrera**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de septiembre de 2017.

Los directores:

Jesús Esteban Díaz Verdejo

Agradecimientos

Poner aquí agradecimientos...

Índice general

Capítulo 1

Introducción

1.1. Motivación y antecedentes

A partir de los años 90 Internet tuvo una gran expansión, pasando de compartir información entre investigadores, como se hacía al principio, a permitir, por ejemplo, realizar compras, videollamadas, pedir citas e informes médicos o gestionar las cuentas bancarias. Internet llega a millones de personas que consumen recursos de forma masiva. Existen múltiples aplicaciones que proporcionan el mismo servicio, por lo que es crucial tener cierta calidad de servicio [1] para mantener una base de usuarios activos.

Como se puede ver en la Figura 1.1, la calidad de servicio es fundamental para que el ancho de banda no se vea afectado por diferentes tipos de tráfico. Si se tienen, por ejemplo, tres tipos de tráfico distinto que ocupan el mismo ancho, al llegar al usuario final no se debe de permitir que uno ocupe todo el canal, cortando los demás tipos de tráfico. Si se está realizando una videollamada, y esta ocupa todo el ancho de banda se tendrá como resultado que otros servicios no funcionen de forma correcta.

Hay que tener en cuenta el tipo de aplicación que se está usando, para saber que prioridad otorgarle. Por ejemplo, para enviar un correo no importa tener que esperar 1 minuto para que se envíe. Pero si se trata del visionado de una película mediante *streaming* y hay retardo, no se cumplirá con los requisitos de calidad de servicio, lo cual ocasionará la perdida de clientes.

Aparte de la calidad de servicio, también hay que tener en cuenta la seguridad, pues se envían y reciben datos de tipo muy sensible, como son los datos bancarios y sanitarios. En España, por ejemplo, la protección de este tipo de datos está regulada por la Agencia Española de Protección de Datos [2] y la Ley Orgánica de Protección de Datos [3].

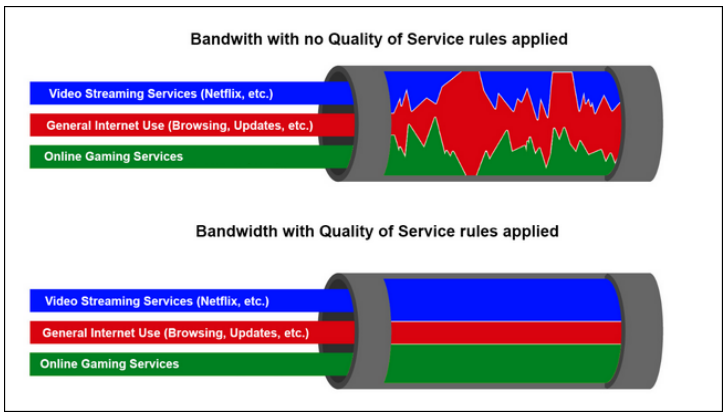

Figura 1.1: Ejemplo del ancho de banda sin calidad de servicio y con calidad de servicio.

Se puede dar prioridad a diferentes tipos de tráfico, mediante su identificación y posterior clasificación. "La identificación de tráfico consiste en asignar instancias de tráfico o elementos a las aplicaciones o tipo de aplicaciones que lo generaron", cita de [4]. Esto puede ser realizado siguiendo tres niveles. A nivel de flujo, a nivel de paquete y a nivel de equipos. Siendo el más común la clasificación a nivel de flujo.

Dentro de las técnicas de identificación de tráfico, la más fiable es la Inspección Profunda de Paquetes, DPI o *Deep Packet Inspection* en inglés[5]. Consiste en analizar los paquetes entrando dentro de su contenido, buscando cadenas que permitan identificar inequívocamente el protocolo. Por ejemplo buscará para *HTTP* las cadenas *GET* y *POST*.

Esta técnica, a pesar de ser la más efectiva cuenta con varios inconvenientes.

- *Escalabilidad.* Ya que se trata de una técnica que tiene que ir paquete a paquete y entrar dentro de ellos requiere de una gran cantidad de recursos. Si se trata de analizar una red con poco tráfico, se obtendrán buenos resultados en cuanto a rendimiento, pero de tratarse de una red con mucho tráfico, se tendrán malos tiempos de análisis.
- *Privacidad.* Al entrar dentro de los paquetes, se produce cierta violación de las políticas de privacidad de la red. Esto puede ser ilegal en algunos países.

Una posible solución parcial a este problema sería la aplicación de la técnica de emparejamiento de flujos [6], desarrollada por investigadores del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada y probada en entornos de laboratorio [7]. Esta técnica además de ser eficiente, respeta la privacidad al no inspeccionar los paquetes de forma profunda.

Por lo tanto, en el presente trabajo se va a implementar esta técnica y se probará más allá de un entorno de laboratorio, llevándola a escenarios reales.

Para llevar esta técnica a un escenario real se precisará de un monitor de redes, NMS, *Network Monitoring System*. El trabajo que realiza es el de monitorizar el tráfico en la red en la que se esté ejecutando. Para este proyecto se usará Bro [8], cuya principal ventaja sobre el resto es la posibilidad de incorporar funcionalidades extras, mediante la creación de módulos.

1.2. Objetivos

El objetivo de este trabajo es el desarrollo de un módulo para un NMS, en este caso Bro. Con el desarrollo del módulo se tratará de demostrar que la técnica de emparejamiento de flujos se puede realizar fuera de un entorno de laboratorio.

Este objetivo se descompone de la siguiente forma.

- Implementación de la función de emparejamiento de flujos, así como el control de los distintos eventos para la gestión del tráfico.
- Gestión de las entradas y salidas. En un principio se hará uso de un archivo *pcap*, de forma que se pueda comprobar que todo funciona correctamente. Una vez terminado será posible realizar una ejecución a tiempo real en una red.
- Realización de pruebas del funcionamiento.

1.3. Metodología

Para realizar este trabajo se establecen una serie de tareas:

- Estado del Arte.
  - Lectura del artículo del departamento. [7]
  - Búsqueda de información sobre la identificación de tráfico.
  - Análisis de las herramientas.
- Diseñar el módulo.
- Implementar el módulo.
- Evaluación y pruebas del módulo.
- Realización de la memoria.

En la siguiente Figura 1.2, se puede ver una temporización de las tareas en forma de diagrama de Gantt.

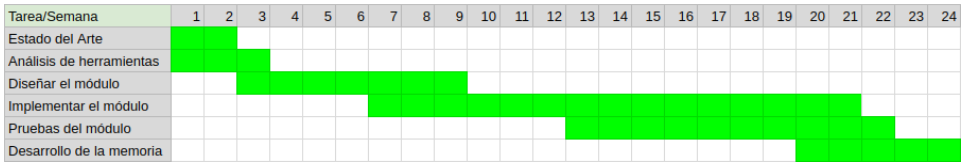

Figura 1.2: Temporización en diagrama de Gantt.

El gasto que requiere este proyecto se desglosa de la siguiente manera.

- *Licencias*. El gasto en licencias será nulo, pues Bro está creado bajo licencia de software libre [8]. El módulo será subido a GitHub [9] con licencia de software libre, por lo que cualquiera podrá usarlo o modificarlo en el futuro.

- *Equipo*. Teniendo en cuenta que la vida útil de un portátil es de unos 4 años y que el desarrollo de este trabajo requiere de unos 6 meses, se tendrá que un portátil de gama media-alta de 800€generará un gasto de un octavo de la vida útil. Por lo tanto será un coste de unos *100€*.

- *Programador*. Según se puede comprobar en Internet [10], el precio por hora de un programador está alrededor de los 30€, por lo tanto si en el desarrollo del proyecto se ha necesitado de unas 500 horas y dejando el precio en 30€por hora, el gasto será de *15000€*.

- *Varios*. Además de los gastos ya descritos, hay que sumar una parte de gastos varios como Internet, luz, agua, y demás. Un montante de 150€al mes, que al ser 6 meses supone un gasto de *900€*.

Teniendo en cuenta todos estos cálculos, se tiene que el coste total del proyecto es de unos *16000€* por 6 meses de trabajo.

1.4. Estructura de la memoria

Esta memoria se organizará de la siguiente forma:

- En el capítulo 2 se explicarán los fundamentos teóricos y tecnológicos sobre los que se basa el presente proyecto.

- En el capítulo 3 se contará cómo se pretende resolver el problema expuesto.

- En el capítulo 4 se encontrará detallado cómo se ha implementado el módulo.

- En el capítulo 5 se realizarán las pruebas, para comprobar que todo funciona como estaba previsto, tanto a nivel funcional como a nivel de aplicación.
- En el capítulo 6 se expondrán las conclusiones recogidas a lo largo de este proyecto y las posibles opciones que existen para seguir trabajando sobre este tema.

Capítulo 2

Estado del arte

En este capítulo se describirán los fundamentos teóricos y tecnológicos del proyecto. En primer lugar se presentará la identificación de tráfico y las distintas técnicas que existen para ello. También se explicará Bro [8], el sistema de monitorización de tráfico que se usará para llevar a cabo el desarrollo del proyecto. Así, se explicará su funcionamiento y, especialmente, el lenguaje de programación incorporado, analizándose cómo gestiona los eventos y sus funcionalidades básicas, así como la posibilidad de ampliar estas.

Por último, se presentará de forma teórica en que consiste el emparejamiento de flujos y de qué forma se podría usar para identificar el tráfico.

2.1. Identificación de tráfico

Una definición de identificación de tráfico podría ser la siguiente, "la clasificación del tráfico implica la asignación de objetos de tráfico a las clases de tráfico que los generan. La identificación usa terminología similar a la clasificación de tráfico. El término identificación se suele usar cuando se realiza de forma granular", cita de [11]. Por lo que, se usará tanto identificación como clasificación indistintamente en las explicaciones de esta memoria.

Esta técnica es usada para realizar muchas tareas de gestión y seguridad de la red. Como por ejemplo medidas de seguridad, garantizar la calidad de servicio [1] e ingeniería de tráfico [4].

2.1.1. Técnicas de identificación de tráfico

Existen distintas técnicas para la identificación de tráfico, siendo distintas en función del nivel de granularidad que se aplique en el análisis [4]. Existen tres grupos, los cuales se presentarán de forma breve.

- *Paquetes.* Se realiza el análisis a los paquetes de forma individual.

- *Flujos*. Se analizan los flujos a partir de ciertos parámetros.
- *Host*. Se identifican las aplicaciones que usan los distintos equipos de la red.

La más usada es la identificación basada en flujos, que es la que se utilizará en este trabajo.

2.1.2. Identificación de tráfico basada en flujos

En esta técnica, los flujos se analizan individualmente, pudiéndose dar un análisis global de la conexión o solamente de algunos de los paquetes que componen al flujo.

Existen tres técnicas básicas de identificación de tráfico basada en flujos.

- Por los puertos de la capa de transporte, establecido por *IANA* [12].
- Por el contenido del paquete o *DPI* [13].
- Por la aplicación de técnicas de aprendizaje automático sobre estadísticas de tráfico, *machine learning* [14].

Estas técnicas serán explicadas de forma más extensa a continuación.

2.1.2.1. Identificación basada en puertos

Esta técnica se basa en identificar el tráfico dependiendo de los puertos de la capa de transporte, según la asignación estándar de IANA [15].

Por lo tanto, IANA es quien asigna el número de puerto oficial a los distintos protocolos, haciendo que, a priori, pueda identificar el tráfico por el número de puerto del servidor.

Esta técnica es la más simple y funcionaba correctamente, pero actualmente no es la más fiable. Esto se debe a la ofuscación de puertos, la multiplexación de puertos por diferentes servicios y el uso de otros puertos no oficiales.

Un ejemplo de multiplexación de puertos podría ser el siguiente. En la actualidad, se están desarrollando multitud de aplicaciones web. Estas aplicaciones se conectan mediante el puerto 80, es decir, mediante el protocolo HTTP. Pero esto es solo teoría, pues se puede hacer que cualquier aplicación envíe información por el puerto 80, sin que pertenezca realmente a HTTP, lo cual daría como resultado una mala identificación.

2.1.2.2. Aprendizaje automático

En esta técnica de identificación, se hace uso de clasificadores basados en algoritmos de aprendizaje automático [14]. Se trata de un campo de investigación muy activo actualmente. En estas investigaciones se han propuesto y evaluado múltiples sistemas basados en distintos tipos de clasificadores, como redes neuronales, redes bayesianas o lógica fuzzy.

A pesar de ser un campo de investigación muy actual y en el cual los métodos implementados son cada vez más inteligentes, los resultados no son buenos. Son costosos computacionalmente y al necesitar tiempo para el aprendizaje, se dan muchos errores en la clasificación.

2.1.2.3. DPI

La Inspección Profunda de Paquetes, DPI por sus siglas en inglés, *Deep Packet Inspection*, realiza un análisis de los paquetes, entrando en el *payload*, en busca de cadenas que permitan identificar de manera inequívoca el protocolo [13]. Dichas cadenas podrían ser *GET* o *POST* para el protocolo *HTTP*, por ejemplo.

Esta técnica suele ser usada por los proveedores de servicios de Internet, ISP, *Internet Service Provider* y grandes empresas. Se puede decir que es una técnica que no respeta la privacidad, pues analiza la información contenida en el paquete. Aunque en el caso de una gran empresa si puede hacerse, pues si se está en la red propia no es delito mirar la información que contienen los paquetes.

A pesar de ser la más efectiva en la actualidad, con una buena tasa de identificación y pocos errores [5], presenta problemas a nivel de escalabilidad, al tener que analizar todos los paquetes de forma individual, y de privacidad, al entrar dentro de los paquetes, pudiendo llegar a ser ilegal en algunos países.

2.2. BRO

Bro [8], es un analizador de tráfico de red de código abierto, por lo que puede ser usado por quien lo desee sin necesidad de pagar licencias. Funciona sobre sistemas basados en Linux y Mac OS X [16]. Una de sus principales características es la gran cantidad de información que puede extraer con un solo escaneo. Otros monitores de red proporcionan menos información, teniendo que ser el propio administrador el que analice después la información obtenida por estos. Por lo tanto, tardará más en resolver los posibles problemas que encuentre en la red.

Figura 2.1: Logo de Bro.

No tiene interfaz gráfica, por lo que su gestión se realiza desde la línea de comandos. Cuando se analiza tráfico, Bro generará unos registros con la información obtenida, los cuales están divididos en función de parámetros definidos por el equipo de desarrollo.

Bro incorpora la posibilidad de introducir funcionalidades nuevas, mediante la programación en su propio lenguaje de *scripting*, del mismo nombre. Esto se expondrá más adelante y, en el capítulo 4, se mostrará cómo se desarrolla y algunos ejemplos de código escrito en Bro.

El lenguaje de *scripting* está orientado a trabajar con eventos. Por lo tanto, a la hora de añadir una nueva funcionalidad habrá que crearla a partir del uso de los eventos que puede gestionar el programa.

Bro está estructurado de forma que todos los flujos de paquetes que analiza son procesados por el motor de eventos, como se puede ver en la Figura 2.2. Este motor convierte los flujos de paquetes en procesos de alto nivel, de forma que es más sencillo trabajar con ellos. Una vez que estos son tratados se generan los registros correspondientes, los cuales podrán ser analizados posteriormente [17].

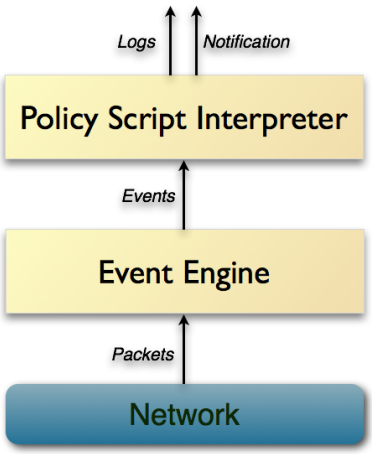

Figura 2.2: Arquitectura de Bro.

2.2.1. Funcionalidades básicas de BRO

La funcionalidad básica de Bro es la monitorización de la red en la que se ejecuta. Mientras que se encuentra en ejecución, genera registros o *logs* en texto plano que se podrán leer usando un editor de texto. Si se analiza un archivo *pcap* los *logs* no cambiarán tras finalizar el procesamiento. Sin embargo, si se analiza tráfico en tiempo real, los registros se irán actualizando a medida que pase el tiempo. Algunos de los *logs* que se generarán son los siguientes.

- *dpd.log*. Consiste en un resumen de los protocolos encontrados en puertos que no son estándar.
- *dns.log*. Contendrá toda la actividad correspondiente al *DNS*
- *ftp.log*. Un registro de la actividad a nivel de sesión de *FTP*.
- *files.log*. Un resumen con los archivos transferidos a través de una red. Incluye protocolos *HTTP, FTP y SMTP.*
- *http.log*. Registro de toda la actividad *HTTP* con sus respuestas.
- *ssl.log*. Un registro de las sesiones *SSL*, incluidos los certificados que se utilizan.
- *weird.log*. En este *log* se guarda la información correspondiente a actividad inesperada o rara a nivel de protocolo. Al analizar gran cantidad de tráfico no es muy útil, pues normalmente considera un volumen importante del tráfico como inesperado, pero a pequeña escala es bastante interesante para detectar, por ejemplo, intrusiones.
- *conn.log*. Aquí se puede ver la información correspondiente a sesiones *TCP, UDP e ICMP*.

Se puede consultar más información sobre *logs* generados por una monitorización de Bro en [18].

Bro dispone además de varios *frameworks* que extienden su funcionalidad. Con ellos se podrán crear *scripts* muy potentes. Algunas de las utilidades más relevantes son las siguientes.

- *Geolocalización*. Se podrá encontrar la localización geográfica de una IP.
- *Análisis de ficheros*. El monitor de red tiene la capacidad de trabajar con ficheros.
- *Framework de loggins*. Con este *framework* se podrá extender los archivos de registro que se generan.

- *NetControl*. Este *framework* permitirá a Bro conectarse con distintos dispositivos de la red, como *switches* o cortafuegos [19]. En la Figura 2.3 se puede ver su arquitectura.

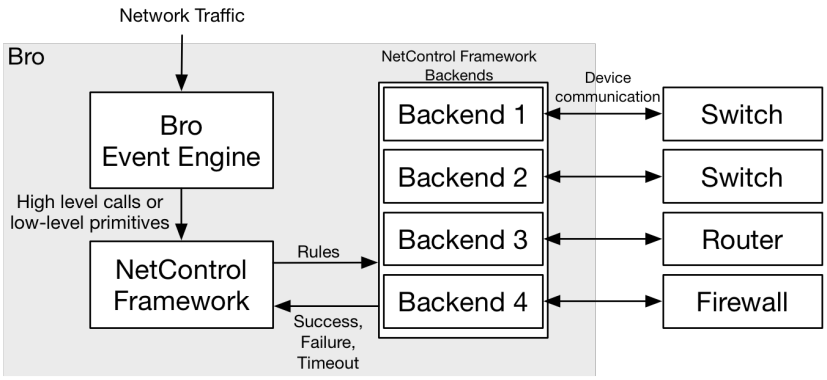

Figura 2.3: Arquitectura de NetControl.

Se pueden ver más detalles de estos *frameworks* y otros en [20].

A partir de la información de los registros, el administrador del sistema podrá determinar, entre otras cuestiones, si existen amenazas en la red o si hay algún componente defectuoso. Para ello deberá hacer uso de los eventos con los que Bro trabaja, para obtener un mayor conocimiento de todo el trabajo que se realice sobre la red.

2.2.2. Eventos y trazas

Aunque Bro permite la creación de funciones, la gestión e identificación del tráfico se realiza mediante eventos. Estos se dan cuando se detecta una determinada acción, por ejemplo, cuando detecta un paquete de respuesta *UDP*, se activará el evento *udp_reply*.

Dentro de cada evento se trabajará con la información del flujo que lo activa. Si captura información de una conexión *TCP*, se tendrá que trabajar con ese tipo de flujo y las distintas variables globales que hayan sido definidas previamente.

Para trabajar será necesario disponer de trazas de red, es decir, capturas de tráfico, que suelen estar en ficheros de formato *pcap*. Se podrán obtener con un monitor de red, siendo en el caso de Bro necesario descargar un módulo adicional llamado *trace-summary* [21]. Además de realizar capturas de tráfico, también da la posibilidad de separar el tráfico entrante del saliente, lo cual generará distintos registros. También se podrán conseguir distintas trazas de la web de Bro.

De todo esto se obtiene que la programación de Bro esta orientada a eventos. Esto supone que no hay programación secuencial, por lo que se ejecutarán los distintos eventos según se vayan activando. Se tendrá que tener en cuenta los eventos que hay disponibles para detectar el distinto tipo de tráfico.

2.2.3. Incorporación de funcionalidades

La incorporación de funcionalidades al monitoreo realizado por Bro es una característica muy llamativa. Gracias a esto se podrá realizar un análisis muy personalizado usando un *script* creado por el administrador de redes. De esta forma podrá, por ejemplo, filtrar el tráfico de una determinada IP mientras se sigue analizando el tráfico de forma normal, con los registros que genera Bro de forma automática. Es una forma muy sencilla de comprobar si por ejemplo el servicio que administra está recibiendo demasiadas peticiones desde una misma IP. Lo cual sería un indicio de ataque de denegación de servicio.

A la hora de incorporar funcionalidades a Bro se puede hacer todo lo que se desee. Una búsqueda rápida por *GitHub* arrojará una gran cantidad de personas que contribuyen con una gran cantidad de nuevas funcionalidades [22]. Ahora lo ideal sería incorporar un módulo, de forma que si el resto de la comunidad lo desea pueda hacer uso de él de una forma sencilla.

2.3. Emparejamiento de flujos

La técnica de emparejamiento de flujos, fue planteada por investigadores del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, en el año 2011 [6] [7], para abordar los problemas de escalabilidad y privacidad de otras técnicas. Hay que tener en cuenta que el emparejamiento de flujos no es una técnica de identificación propiamente dicha, ya que no determina el tipo de flujo. Básicamente esta técnica lo que hace es agrupar los flujos de la misma clase, mediante asociaciones uno a uno.

La idea de la que se parte es que dos flujos próximos en el tiempo, que comparten dirección IP y que usan números de puerto idénticos o próximos, deben de estar relacionados entre sí y corresponderán al mismo protocolo. Esto es, dos flujos que acceden al mismo servidor (IP) y puerto deberían de corresponder al mismo protocolo. Pero también dos flujos del mismo cliente con número de puerto consecutivos y muy próximos en el tiempo corresponderán, muy probablemente, al mismo protocolo y formarán parte de una secuencia de flujos de un interacción.

De esta forma, se define en [7] una función de similitud entre dos flujos a partir de las direcciones IP, los números de puerto y la proximidad temporal

como se puede ver en la ecuación 1:

$$F(x,y) = \begin{cases} G(x,y), & N_{IP}(x,y) \geq 1 \\ -\infty, & \text{en otro caso} \end{cases} \quad (1)$$

Donde G(x,y) es una función que evalúa la semejanza entre dos flujos, como se ve en 2:

$$G(x,y) = |N_{IP}(x,y)–1| + \frac{1}{dp1(x,y)+k1} + \frac{1}{dp2(x,y)+k1} + \frac{1}{dt(x,y)+k2} \quad (2)$$

Las variables de la función son las siguientes:

- *x, y*: Primer paquetes o flujos a comparar.
- $N_{IP}(x,y)$: Número de IP's coincidentes en ambos paquetes o flujos, estando el valor comprendido entre 0 y 2.
- *dp1(x,y)*: Se corresponde con la diferencia entre los números de los puertos de origen de los dos paquetes.
- *dp2(x,y)*: Será la diferencia entre los números de los puertos de destino de los dos paquetes.
- *k1, k2*: Son constantes que deben de ser estimadas experimentalmente. En [7] se proponen valores entre 0.1 y 10000.
- *dt*: Es la diferencia de tiempo existente entre los tiempos de inicio de los flujos (*timestamps*).

El emparejamiento se realiza a partir del valor de similitud obtenido mediante la comparación con un umbral, que debe ser ajustado experimentalmente. Si se intentan emparejar todos los flujos mediante un umbral bajo, se producirán muchos errores, esto es, habrá una tendencia a que se consideren iguales flujos que no lo son, ya que pasarán el corte del umbral.

Como se ha mencionado, el emparejamiento de flujos, por si mismo, no identifica el tráfico. Por lo tanto se necesitará otra técnica para clasificar el tráfico.

Capítulo 3

Diseño y arquitectura del sistema

En este capítulo se abordará el diseño del sistema. Para ello, a partir del estudio de los métodos y procedimientos disponibles en Bro para la gestión de flujos, se determinarán los módulos y funcionalidades necesarias, proponiéndose una arquitectura para el sistema a implementar.

Así, en primer lugar se presentará la arquitectura propuesta y los diferentes módulos y funcionalidades. También se describirán las estructuras de datos usadas para la gestión de la información.

3.1. Arquitectura del sistema

Para describir la arquitectura del sistema hay que tener en cuenta la arquitectura de Bro. Este monitor de red es un software modular, esto es, esta compuesto de diferentes módulos que al ser ejecutados funcionan como un único sistema.

Por lo tanto, la arquitectura del sistema a desarrollar se debe de acoplar a la arquitectura propia de Bro, por lo que el sistema debe implementarse como un módulo adicional compatible con Bro. Entre los requisitos del mismo se encuentra que sea ligero y eficiente. Por lo tanto, se deberán usar los distintos eventos y capacidades que proporciona Bro para minimizar el impacto en el sistema global y optimizar su funcionamiento.

Se prescindirá del uso de los *frameworks* descritos en el capítulo anterior (2.2.1), pues su uso no aporta nada relevante que no se pueda realizar exclusivamente con los eventos ya disponibles destinados a gestionar el tráfico de la capa de transporte. Así, se propone usar este tipo de eventos como núcleo y soporte del módulo y de todas las funcionalidades necesarias. Las dos funcionalidades más relevantes del módulo están relacionadas con la evaluación de la similitud entre flujos y la gestión de las listas de flujos en

diferentes situaciones. Así, se definirá una función que se encargará de evaluar la fórmula del emparejamiento de flujos (Apartado 2.3). Esta función devolverá un número que será el que se compare con el umbral.

En la Figura 3.1, se pueden ver los distintos pasos que se seguirán en la ejecución del módulo, siendo estos complementados por los pasos que se ven en las Figuras 3.5 y 3.6, de la siguiente sección.

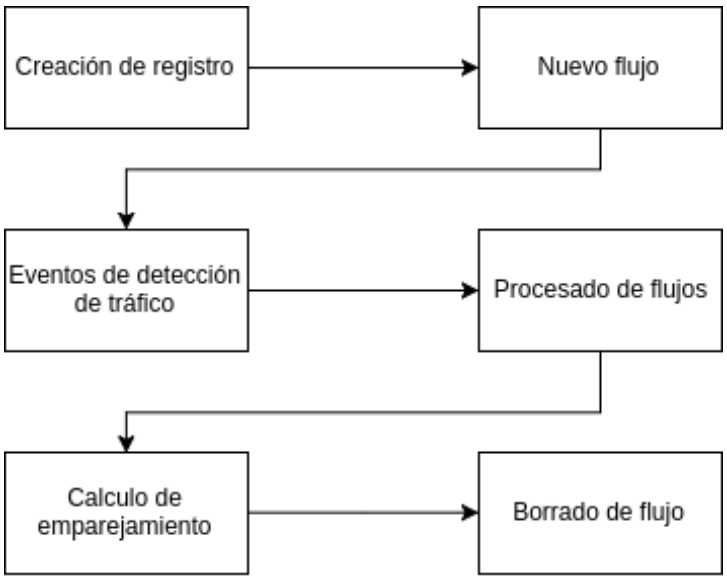

Figura 3.1: Diagrama del sistema.

De esta forma, se creará un registro, en el cual se almacenarán los flujos que son emparejados. De esta forma se evita tener que iterar sobre la tabla de emparejados al finalizar el módulo, haciendo que sea más eficiente.

En la Figura 3.2, se puede ver cómo se procederá a analizar un flujo cuando es detectado. En primer lugar se detectará el flujo y se iniciará la búsqueda de otro con sus características, en caso de que se encuentre, se comenzará con el análisis. Si no se detecta ningún flujo con sus parámetros, se almacenará.

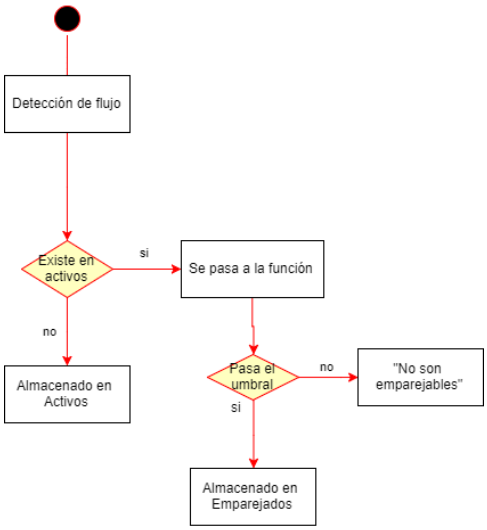

Figura 3.2: Detección de un nuevo flujo.

En la Figura 3.3, se puede ver cómo se comportará el sistema cuando vaya a eliminar un flujo de la memoria. Se buscará un flujo en la lista de activos para que lo reemplace, en caso de que no haya ninguno con sus características se borrará de la memoria.

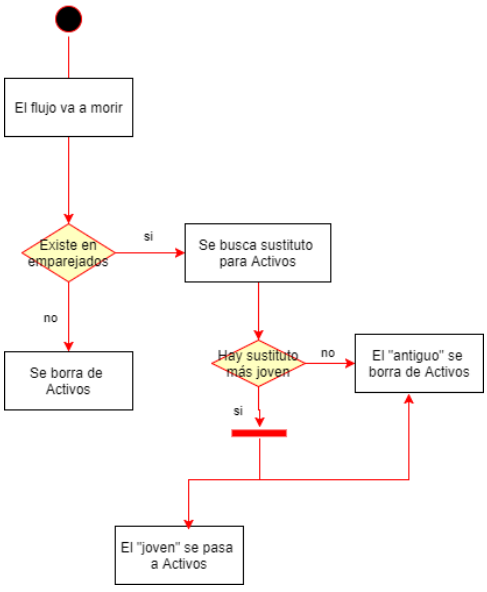

Figura 3.3: Muerte de un flujo.

Dependiendo del trato que se les vaya a dar a los flujos detectados, se almacenarán en una lista o en otra. Estos pueden estar en diferentes estados,

siendo estos los que se consideran:

- Activo. El flujo está activo y almacenado.
- Emparejado. El flujo ha sido emparejado con otro flujo activo.
- Finalizado. El flujo ha cumplido su tiempo de vida y es borrado de la memoria.

Por lo tanto, cuando un flujo es detectado tendrá el estado activo. En función de los distintos flujos que se vayan detectando los que están en estado *activo* se irán comparando con los nuevos que se detecten, de modo que los nuevos pasarán a estar emparejados. Si no se encuentra un flujo activo que coincida con sus parámetros, los nuevos flujos serán almacenados como activos. Al último estado, finalizado, se podrá pasar tanto del estado activo, como del emparejado, con la diferencia de que, de tratarse del primer caso, deberá de ser borrado de la lista y se buscará un sustituto entre los emparejados con ese flujo. En el segundo caso no se borrará de la lista. La transición de estados se puede ver en la Figura 3.4.

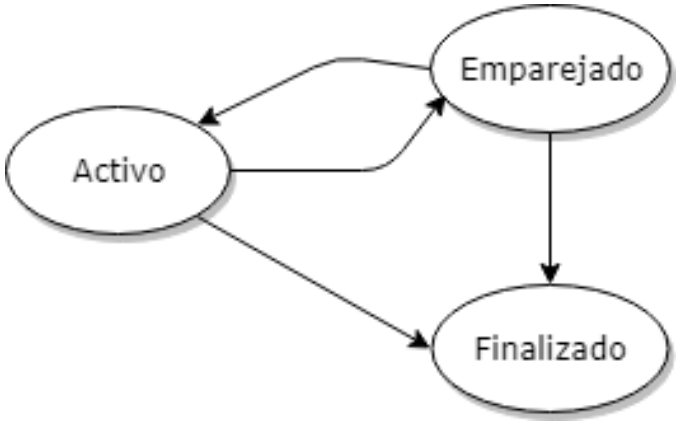

Figura 3.4: Distintos estados de los flujos.

Para el análisis, será necesario disponer de tráfico. Este entrará en forma de traza con formato *pcap*. Las salidas se darán en forma de registros, en los cuales se mostrarán los flujos que han sido emparejados.

La configuración de Bro, se realiza mediante la línea de comandos, cuando se va a lanzar el programa. Por lo tanto, para el módulo que se está describiendo es preciso únicamente activar la opción *-r*, de forma que se le permita leer el archivo que se le pasará como parámetro a continuación. En el caso de que se quiera escanear el tráfico de una interfaz, se deberá de activar la opción *-i* indicando a continuación el nombre de la interfaz a analizar. Se puede ampliar esta información leyendo la ayuda de Bro con *-h* o en el apéndice B.

3.2. Módulo y funciones

Las funcionalidades que se espera que tenga este módulo en esencia son dos, la detección y almacenado del tráfico, y la aplicación de la fórmula para conocer si dos flujos son emparejables. Aunque estas son las dos funcionalidades básicas, también será necesario disponer de algunas funcionalidades extra. De una forma más amplia las funciones del módulo serán las siguientes.

- *Función que aplique la fórmula de emparejamiento.*
  A esta función se le pasará dos flujos, de forma que se aplique la fórmula y devuelva un número, el cual será el que indique si los flujos son emparejables o no.

- *Función de preprocesado.*
  Esta función se encargará de indicar si dos flujos son candidatos a ser emparejados o no. Si son candidatos, se llamará a la función de emparejamiento, de forma que con el resultado que devuelva se decida si se almacenan como emparejables o no.
  En el uso de esta función se almacenará o no el flujo que está siendo analizado. Por lo tanto será necesario el uso de algún tipo de contenedor para este cometido.

- *Funciones que detecten el tráfico.*
  Esto se hará con los eventos de Bro. Los eventos detectarán el tipo de tráfico que se está analizando y llamarán a la función anterior.

- *Creación de registro.*
  A su vez, será necesario crear un registro o *log* con la información que aporta el sistema de identificación.

En la Figura 3.5, se puede ver cómo se realizará el preprocesado de un flujo antes de aplicar la ecuación de emparejamiento 2.

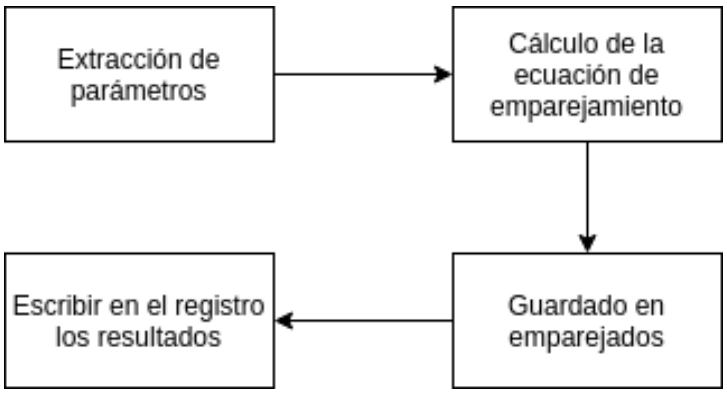

Figura 3.5: Diagrama de la función previa a la aplicación de la ecuación de emparejamiento.

En la Figura 3.6, se ve cómo se tratan a los flujos que han pasado el preprocesado y que son candidatos a ser emparejados.

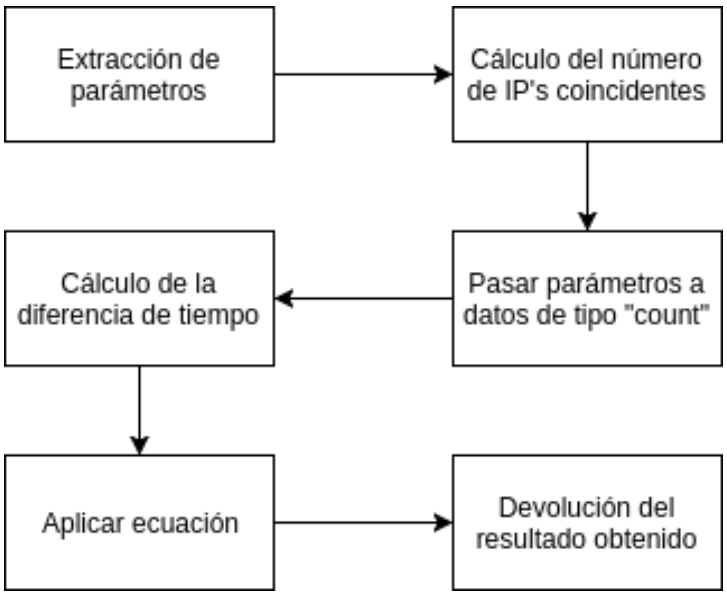

Figura 3.6: Diagrama de aplicación de la función de emparejamiento.

Lo que se espera es capturar el tráfico de la capa de transporte. Dicho tráfico se corresponde a los protocolos *TCP y UDP*. Por lo tanto será necesario ver que tipo de eventos son los que controlan el tráfico de estos dos protocolos. A su vez, se tendrá que crear variables globales para el almacenamiento de los flujos que sean emparejables y los que estén activos.

Las entradas y salidas son de fácil gestión. Las entradas de tráfico podrán ser mediante archivos o analizando directamente el tráfico de la red. Las

salidas, por su parte, se realizarán mediante registros. Con esto, se guardará en un fichero en la ruta donde es lanzado el monitor, siendo el posterior análisis mucho más cómodo desde, por ejemplo, un editor de texto.

Se debe de tener en cuenta que siempre se podrá extender la funcionalidad del módulo. Pero de momento no resulta interesante. La posible extensión correspondería a posibles trabajos futuros.

Para realizar este módulo es necesario conocer como gestiona los flujos Bro y de que forma se mantendrán los que son emparejados y los que están activos.

3.3. Gestión de flujos

La gestión de los flujos se realiza completamente mediante los eventos de Bro.

Existen eventos que gestionan el nacimiento y la muerte de los flujos, siendo estos dos eventos genéricos. Por el contrario, se deberán de usar los distintos eventos especializados para gestionar los distintos tipos de tráfico.

De esta forma, se tendrán eventos únicos que detectarán el tráfico *TCP* y otros para *UDP*. Una vez dentro, se realizarán los pasos necesarios para llevar el flujo detectado hasta la función de evaluación.

En el evento correspondiente al nacimiento de un flujo, se deberá de analizar si se tiene otro con el que compararlo, o directamente se almacena con los demás flujos activos.

En cuanto al evento relacionado con la muerte de un flujo, se tendrá que evaluar si en la lista de emparejados se tiene otro, al cual le quede más tiempo de vida, para que lo reemplace en la lista de flujos activos.

A continuación se verán las estructuras de datos necesarias para llevar a cabo el desarrollo del módulo.

3.4. Estructuras de datos

Las principales estructuras de datos que se necesitan para el desarrollo de este trabajo serán dos tablas, *table* [23], de vectores para el almacenamiento de los flujos activos y los emparejados. Los vectores son iguales que en cualquier otro lenguaje de programación, mientras que las tablas son parecidas a los *maps* de *C++*. Aunque esto se explicará más detalladamente en el capítulo 4.

Dichas estructuras de almacenamiento, deberán de ser capaces de estar ordenadas por las IP's y los puertos. Esto se obtiene haciendo que las tablas estén compuestas de vectores, por lo cual, se obtendrá una matriz bidimensional indexada. Por lo tanto se consigue que el acceso a los datos sea mucho

más rápido. Incluso se podría prescindir de bucles, los cuales pueden acabar siendo un problema en cuanto a rendimiento si se analiza mucho tráfico. En la Figura 3.7, se muestra cómo están organizadas las matrices.

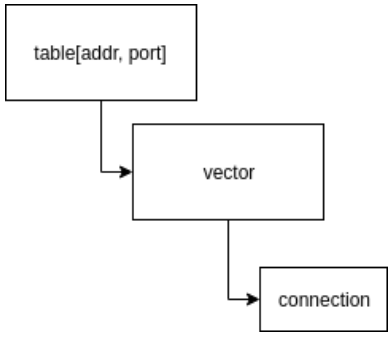

Figura 3.7: Organización de las tablas.

Bro proporciona cierto tipos de datos muy interesantes, los cuales además, incluyen mucha más información. Algunos de estos tipos de datos son.

- *connection.*

  Este tipo de dato es el flujo en sí. Por lo tanto será de vital importancia comprenderlo para poder trabajar como se desea.

  Dentro de *connection* existe un registro llamado *id*, el cual esta compuesto por el tipo de dato *conn_id* [24]. Este dato sirve para identificar los flujos mediante una tupla formada por 4 datos. Estos datos son los que se precisan para indexar la matriz bidimensional. En la Figura 3.8 se muestran los dos datos que se usarán de *connection.*

  - *addr.* Este tipo de dato representa una IP. Reconoce tanto IPv4 como IPv6. Este tipo de dato puede ser comparado e incluso ordenado mediante operadores [25].

  - *port.* Este tipo es el usado para los puertos. Además del número, también indica el protocolo de la capa de transporte que usa. Soporta la comparación y ordenación, pero no por el número, sino por el tipo de protocolo [26].

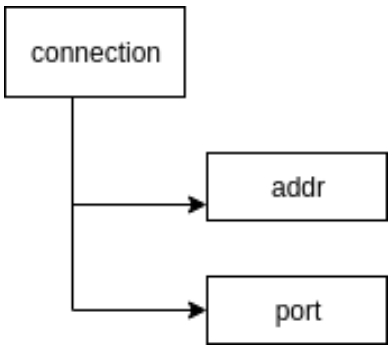

Figura 3.8: Tipo connection desglosado.

Se puede consultar más información sobre el tipo de dato *connection* en [27].

- *time*.

  Aunque en otros lenguajes se pueden crear estructuras que se asemejen a este tipo, Bro da un tipo de dato muy completo. Además, se puede operar sobre él desde el principio, siendo una gran ventaja a la hora de calcular el tiempo de inicio de los flujos. Más información en [28].

  Es importante entender que para realizar el cálculo para el emparejamiento de flujos, se necesita el *timestamp* del primer paquete de cada flujo, pues será sobre este tiempo sobre el que se apoye el cálculo del emparejamiento.

Estos dos tipos de datos a parte de ser los más interesantes para el cálculo del emparejamiento, también serán los más utilizados junto a los contenedores para los flujos. Existen más tipos de datos e incluso los hay que extienden la información disponible sobre los flujos. Más información sobre distintos tipos de datos en [29].

Capítulo 4

Implementación

A continuación, se expondrá cómo se ha implementado el módulo de Bro, el cual sirve como resolución al problema planteado en 2.3.

La descripción del módulo se realizará mostrando las cabeceras de las funciones y eventos empleados, comentando después para que sirve cada uno. De forma que se tendrá una descripción de la implementación a modo de *API*.

4.1. Creación del registro

La creación del registro sobre el que se escribirán los flujos que son emparejables, se realiza mediante un evento.

```
1 event bro_init()
```

Este evento es lanzado cuando Bro se inicia. Cuando se inicie se creará un registro, en el que se escribirán las IP's de origen y destino, así como los puertos. Además, se el *uid* de cada flujo, para poder realizar búsquedas rápidas. También, entre la información de los flujos habrá un string, cuya función es separarlos para una mejor visibilidad.

4.2. Detección de flujos nuevos

A continuación, se explicará el evento encargado de gestionar los flujos de nuevos.

```
1 event new_connection(c: connection)
```

Este evento recibe como entrada una nueva conexión, la cual no está identificada previamente. Esto quiere decir que es nueva o que ha sido borrada de la memoria y ha vuelto a ser detectada.

Una vez que se detecte el flujo, se deberá de comprobar si existe, de forma que se descartará, o si no existe, por lo que se creará un nuevo índice en la tabla de activos y se almacenará.

Este tipo de evento detecta las conexiones de tipo *TCP y UDP*.

4.3. Eventos de detección de tráfico

A continuación, se verán los distintos eventos que van a detectar el diferente tipo de tráfico.

```
1 event connection_established(c: connection)
2
3 event connection_finished(c: connection)
4
5 event udp_request(u: connection)
6
7 event udp_reply(u: connection)
```

Los dos primeros eventos son los correspondientes al tráfico *TCP*. Los otros dos, como se indica en el nombre están destinados al tráfico *UDP*.

El primer evento relacionado con *TCP*, se activa cuando se detecta un paquete *SYN-ACK* que responde al *handshake* que se realiza en las conexiones de este tipo.

El segundo evento detecta cuando la conexión *TCP* finaliza de forma normal.

Los dos eventos relacionados con *UDP* detectan paquetes de dos tipos distintos.

- *UDP request*. Se genera por cada paquete que es enviado por el creador del flujo.
- *UDP reply*. Este es generado por cada paquete que es enviado por el receptor del flujo.

Estos dos últimos eventos son bastantes costosos, pero son absolutamente necesarios, ya que son los únicos eventos que detectan este tipo de conexión.

Dentro de todos estos eventos se realiza el mismo proceso. Primero se obtienen las IP's y los puertos del flujo y se obtiene el flujo activo correspondiente a ese flujo. Después de esto, se comprueba que IP y que puerto coinciden, llamando así a la función de *calculo* para que realice el filtrado.

Aparte de estos eventos, también existen dentro del módulo otros dos eventos, destinados a detectar paquetes del protocolo *ICMP*. Estos eventos son necesarios, pues Bro los detecta igual que los de tipo *TCP y UDP*, aunque no pertenezca a la capa de transporte.

```
1 event icmp_echo_request(c: connection, icmp: icmp_conn, id: count, seq
      : count, payload: string)
2
3 event icmp_echo_reply(c: connection, icmp: icmp_conn, id: count, seq:
      count, payload: string)
```

El protocolo *ICMP* se usa para el control, enviando mensajes de error en caso de que un router o un host no sean accesibles.

Al igual que con los eventos de *UDP*, el *request* es enviado por el creador del flujo, siendo una petición. El *reply* es enviado por el receptor del *request*, por lo que se considera la respuesta del anterior. El funcionamiento es el mismo que en los anteriores eventos que gestionan el tráfico.

4.4. Procesado de flujos y escritura en el registro

Ahora, se va a describir cómo se procesan los flujos antes de ser enviados a comparar. A su vez, se hablará de cómo se escribe en el registro.

```
1 function calculo(c1: connection, c2: connection )
```

Esta función se usa para procesar los flujos antes de llamar a la función de emparejamiento. Aquí, se comprueba primero que no se está analizando el mismo flujo. Una vez que esto ha sido descartado, se llama a la función de emparejamiento, y con el número que devuelve, se comprueba si son emparejables o no. Si se obtiene un resultado mayor que el umbral, serán emparejables, y por lo tanto se almacenarán en la tabla de emparejados. En caso contrario, no se almacenarán en la tabla de emparejados.

Además de lo ya detallado, cuando son emparejables, se escribirá la información de los dos flujos en el registro.

4.5. Cálculo de emparejamiento

A continuación, se explicará cómo se aplica la ecuación 2 en el módulo.

```
1 function emparejamiento(c1: connection, c2: connection ):double
```

Esta función recibe como entrada dos flujos y devuelve un número, de tipo *double*.

Lo primero que se hace es sacar las IP's de origen y destino y los puertos de origen y destino de los flujos. También, se obtendrán los *timestamps* de los flujos, de esta forma se conseguirá la diferencia de tiempo.

Para poder operar con los puertos, habrá que pasarlos al tipo *count*, de esta forma se elimina la terminación con el tipo de protocolo del puerto.

Se puede operar con esta terminación, pero al tener que usar después otros tipos de datos que no son puertos es mejor quitar la terminación, pues de lo contrario se obtendrá un resultado erróneo.

Lo mismo que con los puertos pasa con el tipo *time*. Se puede operar con este tipo de dato pero no es recomendable hacerlo ya que se va a trabajar con otros datos de carácter matemático.

El número de IP's coincidentes en ambos flujos, o N_{IP} en la fórmula, es de fácil cálculo, pues bastará con saber si coinciden una IP o las dos.

4.6. Borrado de flujos

Ahora, se describirá el funcionamiento del borrado de flujos.

```
1 event connection_state_remove(c: connection)
```

Este evento se activa cuando el flujo que entra como parámetro va a morir, o ser borrado de la memoria. Es un flujo que ya ha sido procesado por el módulo.

Lo que se realiza dentro de este evento es buscar en la tabla el índice el vector. De esta forma se borrará el primer flujo almacenado en el vector.

De ser el único flujo almacenado en el vector, se borrará el vector entero. Si hay más flujos almacenados, se moverán una posición hacia atrás. De esta forma, se seguirá teniendo un rendimiento óptimo al no tener otra tabla que indique que flujos son borrados.

4.7. Almacenamiento

Por último, se verá cómo se ha implementado las estructuras de almacenamiento de los flujos.

```
1 global activos: table[addr, port] of vector of connection;
2
3 global emparejados: table[addr, port] of vector of connection;
```

Se trata de dos tablas globales, cuyo indice está constituido por una IP, ya sea la de origen o la de destino, un puerto. Además, como los indices son únicos, cada índice apuntará a un vector, y dentro de este se almacenarán los flujos ordenados, en función de cuando son detectados.

La primera tabla será usada para almacenar los flujos que están activos, por lo tanto el primer flujo de cada vector será el usado para realizar las comparaciones. La segunda se empleará para almacenar los que ya están emparejados, por lo que contiene flujos que pueden haber sido borrado de memoria.

Capítulo 5

Evaluación y pruebas

En este capítulo se abordará la prueba del sistema. Para ello, a partir de un archivo que contenga una traza de tráfico, se procesará con el módulo creado, de forma que se compruebe que se crea el registro correspondiente y que este contiene los flujos emparejados.

Se añadirán capturas de pantalla en las que se vea claramente que todo funciona como debería, así mismo, se podrán comprobar los registros que se creen en el repositorio de GitHub [9].

5.1. Establecer las variables

Lo primero que se hará, será establecer las variables *k1* y *k2*. Estas se definen de forma global al principio del módulo. Cómo ya se vio anteriormente, ecuación 2, sus valores estarán comprendidos entre 0,1 y 10000. Como se puede ver en la Figura 5.1, para las pruebas se han establecido los valores en 1 y 100.

```
## Variables para el calculo de la ecuacion
global k1 = 1;
global k2 = 100;
```

Figura 5.1: Valores de las variables *k1 y k2*.

A su vez, también será necesario establecer el umbral que servirá para descartar o aceptar emparejamientos. Como se puede ver en la Figura 5.2, en el caso de estas pruebas el valor será de 1.

```
## Definimos el umbral,
global umbral=1;
```

Figura 5.2: Valor de la variable *umbral*.

5.2. Prueba del módulo

A continuación, se realizará una prueba del módulo, con las variables anteriormente descritas, para comprobar que todo funciona correctamente.

Como se puede ver en la Figura 5.3, para lanzar el módulo, será necesario indicar el archivo que se va a analizar.

```
alvaro@alvaro-IdeaPad-U410 ~/Escritorio/bro $ bro -b -r pcap/nitroba.pcap script
s/bro-flows/broflows.bro > ~/Escritorio/prueba3-conlog.log
```

Figura 5.3: Lanzamiento del módulo para realizar el análisis.

En el registro que se crea cada vez que se ejecuta el análisis, además de las IP's y los puertos, también se muestran los *uid* de cada flujo, este parámetro consiste en un identificador único de flujo. Esto ayudará a identificar los flujos que son emparejados, y además servirá para descartar errores en el análisis.

En la Figura 5.4, se puede comprobar el registro que se genera al analizar un archivo. En este caso el archivo analizado a sido *nitroba.pcap*, cuyo peso es de 56,2 MB, el cual se encuentra accesible desde la web de Bro. Esta traza se corresponde con un escenario de prueba realizado por una universidad sudafricana.

broflows.log

```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   broflows
#open   2017-09-10-14-32-59
#fields orig1   po1 dest1   pd1 uid1    informacion orig2   po2 dest2   pd2 uid2
#types  addr    port    addr    port    string  string  addr    port    addr    port    string
69.22.167.201   80  192.168.1.64    46311   CAs2LRdAnuYloVafh    emparejado con     69.22.167.201   80
192.168.1.64    43109   CON5Tk2DEvtktzfQNe
69.22.167.201   80  192.168.1.64    46311   CAs2LRdAnuYloVafh    emparejado con     69.22.167.201   80
192.168.1.64    42271   CaKWe32OxbaQuLFvL
24.64.134.214   18236   192.168.1.64    1026    CrJwxn4RNjL4nzF0nk   emparejado con     24.64.134.214
18236   192.168.1.64    1027    C9Ak9r31rMDGsJQr73
24.64.134.214   18236   192.168.1.64    1026    CrJwxn4RNjL4nzF0nk   emparejado con     24.64.134.214
18236   192.168.1.64    1028    CZCljA4D6W9SXgBDR9
24.64.150.46    2960    192.168.1.64    1026    CqJcV23ACiq7uiiMgf   emparejado con     24.64.150.46
2960    192.168.1.64    1027    CUKNIT1kUROzhfHaa3
24.64.150.46    2960    192.168.1.64    1026    CqJcV23ACiq7uiiMgf   emparejado con     24.64.150.46
2960    192.168.1.64    1028    CGOWNEtxnrCCThacg
192.168.15.1    53  192.168.15.2    32035   CJiwzg3ja9NzKoVgN    emparejado con     192.168.15.1    53
192.168.15.2    11742   CK0cD54uqFor8PbqKl
192.168.15.1    53  192.168.15.2    32035   CJiwzg3ja9NzKoVgN    emparejado con     192.168.15.1    53
192.168.15.2    53752   Cqdv2DK9ltbSsWKck
24.64.79.171    22828   192.168.1.64    1027    C0L1AR2ca6kxWmyW79   emparejado con     24.64.79.171
22828   192.168.1.64    1028    CvnBb84x9kvSvAfP2k
24.64.79.171    22828   192.168.1.64    1027    C0L1AR2ca6kxWmyW79   emparejado con     24.64.79.171
22828   192.168.1.64    1026    C6Ksfkd3gytigNHp5
222.161.2.46    51227   192.168.1.64    1027    CKCXvd4qnDkyK6UOj    emparejado con     222.161.2.46
51227   192.168.1.64    1026    CL6tkFiRsbI9QPVAh
66.179.217.49   80  192.168.15.4    34780   CfgvnD4wKlPly0bZMi   emparejado con     66.179.217.49   80
192.168.15.4    34784   CU388b31CMQWJbnUUj
66.179.217.49   80  192.168.15.4    34780   C1T98A28Y9caRivOMk   emparejado con     66.179.217.49   80
192.168.15.4    34784   CkJUybdb5LAJT3uad
66.179.217.49   80  192.168.15.4    34800   C4N3xXskQoBZRZ9Ek    emparejado con     66.179.217.49   80
192.168.15.4    34806   C2TS3b3P02VeY5kdSc
```

Figura 5.4: Log del archivo *nitroba.pcap*.

Como se puede ver en 5.4, los flujos son emparejados, por lo tanto se puede decir que el módulo funciona correctamente, al menos con trazas

pequeñas. Este tipo de archivos son ideales para realizar las pruebas, pues tarda muy poco el sistema en analizarlos, siendo en este caso 1 segundo el tiempo utilizado.

Además, no se entra dentro de los paquetes, por lo que la privacidad está garantizada.

5.3. Escenarios reales

Una vez que se ha comprobado que todo funciona como debe con un archivo pequeño, se pasará a analizar un archivo más grande, correspondiente a un escenario real. En este caso se trata de *maccdc2012_00004.pcap*, correspondiente al *National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition* o *MACCDC* [30]. En este se encuentra el tráfico de un día y ocupa cerca de 1.1 GB.

En la Figura 5.5 se puede ver que se trata de un registro mucho más extenso que el anterior. De hecho, tarda cerca de 50 minutos en realizar el análisis, frente al segundo que se tardaba con el anterior archivo.

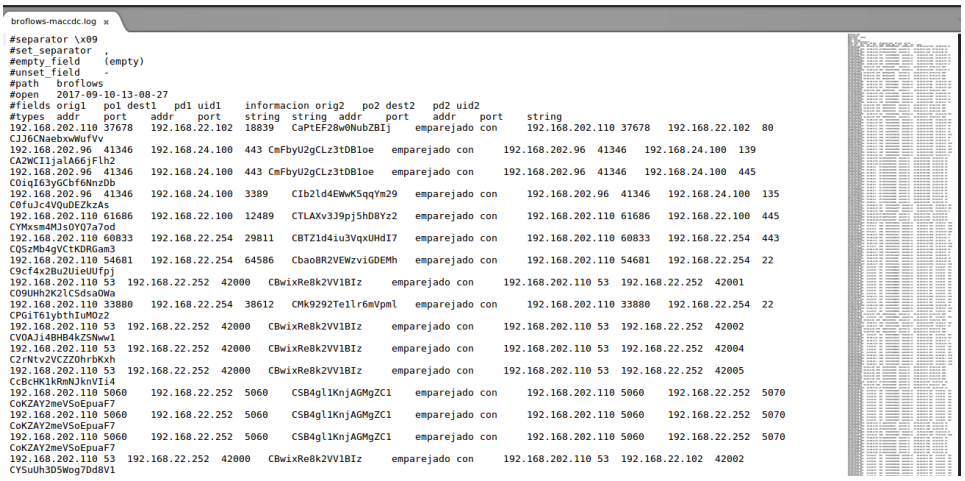

Figura 5.5: Log del archivo *maccdc2012_00004.pcap*.

En la Figura 5.6, se puede comprobar cómo está almacenado distintos flujos, los cuales están emparejados con el mismo flujo activo, por lo que se tiene que todos esos flujos pertenecerán al mismo. Esta comprobación se puede realizar mediante los *uids*, siendo el mismo para el primer flujo, o flujo activo, y distintos en los siguientes, por lo que van a IP's distintas o a puertos distintos, pero pertenecen al mismo flujo general de datos.

```
50 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.24.100 445
   CZDGWc1WCt4zUzwfdd
51 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.23.100 445
   C8BeEggQNQeP4n8Ii
52 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.21.100 445
   CvCEDK1c9dsIRJK0wj
53 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.21.102 445
   CbL4cu3D5ltSkq1Xvi
54 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.21.103 445
   CEasau1qY4VimI2hih
55 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.24.202 445
   CsNBhr1WYQMVltYjbj
56 192.168.202.108 62361 192.168.28.1 443 C98isE12hXh8k5eQT6 emparejado con 192.168.202.108 62361 192.168.21.202 445
   CDG2lf8N7hyzopnHf
```

Figura 5.6: Flujos emparejados.

Si se realiza una búsqueda usando el *uid* del primer flujo, se verá que no se muestra más veces, por lo que se realiza el borrado de la tabla de activos correctamente.

Capítulo 6

Conclusiones y trabajo futuro

Por último, se expondrán las conclusiones a las que se han llegado, así como el posible trabajo futuro a partir de lo realizado.

6.1. Conclusiones

Tras todo lo expuesto a lo largo de esta memoria, y tras ver los resultados en el capítulo 5, se llega a la conclusión de que la técnica propuesta por los investigadores del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, es una técnica completamente funcional. Además, se trata de una técnica que respeta la privacidad y completamente escalable.

Respeta la privacidad, por que no entra dentro del payload de los paquetes. A su vez, es escalable, cómo se ha visto, pues aunque el archivo a analizar sea mucho mayor, los tiempos de análisis son aceptables con respecto al tamaño. Además, no sufre de ningún tipo de retardo.

Los datos mostrados en los registros que se generan, son útiles para realizar una búsqueda rápida en el caso de que sea necesaria y, además, con un clasificador adecuado la tarea de la clasificación será muy llevadera.

6.2. Trabajo futuro

En el futuro se podría seguir trabajando en este módulo, añadiendole funcionalidades que de momento no tiene, por ejemplo, se podrían agregar más eventos que detecten tráfico *TCP*, pues Bro proporciona bastantes de este tipo.

A su vez, podría identificarse tráfico de la capa de aplicación, simplemente añadiendo los módulos adecuados de Bro. Por lo tanto, hay mucho

posible trabajo que hacer.

Apéndice A

Instalación de Bro

En este apartado se van a detallar los pasos para instalar Bro en un sistema Linux.

A.1. Descargar Bro

Los binarios de Bro se pueden descargar desde la propia web de Bro [16], o desde su repositorio de GitHub.

A continuación, se recomienda seguir los pasos que nos detallan en su web [31], aunque serán detallados aquí.

Lo primero será decidir de donde descargar los binarios, se recomienda hacerlo desde el repositorio de GitHub, pues así será mucho más sencillo tenerlo actualizado. Para ello, se realizarán los siguientes pasos:

```
sudo apt-get install cmake make gcc g++ flex bison libpcap-dev libssl-
    dev python-dev swig zlib1g-dev
```

De esta forma, se obtendrán los paquetes necesarios para que Bro funcione correctamente. A continuación, clonaremos el repositorio de GitHub de Bro, de la siguiente forma:

```
git clone --recursive git://git.bro.org/bro
```

A.2. Instalación

Para el proceso de instalación, será necesario, antes de ejecutar los comandos siguientes, estar en la carpeta *bro* que se ha generado con la descarga. Una vez que se esté en dicha carpeta, se procederá a la instalación con los siguientes comandos:

```
./configure
make
make install
```

Una vez realizados estos pasos, se tendrá Bro instalado y listo para usarse.

Apéndice B

Uso del módulo creado

En este apartado se encontrará una pequeña guía de los comandos de Bro, así como una pequeña guía del uso del módulo creado.

B.1. Comandos de Bro

A continuación, se encontrará una lista de los comandos más interesantes de Bro.

```
    -b|--bare-mode                  | no carga los scripts del
        directorio base/
    -d|--debug-policy               | activa el modo debug
    -f|--filter <filter>            | filtro tcpdump
    -h|--help|-?                    | ayuda
    -i|--iface <interface>          | lee de la interfaz dada
    -r|--readfile <readfile>        | lee el archivo dado
    -s|--rulefile <rulefile>        | lee las reglas del archivo dado
    -t|--tracefile <tracefile>      | activa la ejecucion de trazas
    -w|--writefile <writefile>      | escribe en el archivo dado
    -x|--print-state <file.bst>     | muestra el contenido del archivo
        de estado
    -C|--no-checksums               | ignaora checksums
    -F|--force-dns                  | fuerza DNS
    -N|--print-plugins              | muestra los plugins disponibles y
         sale
    -Q|--time                       | muestra el tiempo de ejecucion en
         stderr
    -R|--replay <events.bst>        | repite los eventos
    -X|--broxygen <cfgfile>         | genera la documentacion basada en
         el archivo de configuracion dado
```

B.2. Uso de Bro

Para usar Bro se tendrá que hacer lo siguiente.

```
       bro [opciones] [archivos]
```

Pero antes de la ejecución de Bro, se tendrá que fijar el *PATH* en la carpeta de Bro.

```
export PATH =/usr/local/bro/bin:$PATH
```

Una vez realizado este paso, se podrá ejecutar el módulo creado, de la siguiente forma.

```
bro -b -r pcap/nitroba.pcap scripts/bro-flows/broflows.bro
```

Bibliografía

[1] Microsoft. Calidad de servicio. URL https://msdn.microsoft.com/es-es/library/hh831679(v=ws.11).aspx.

[2] AEPD. Agencia española de protección de datos. URL https://www.agpd.es/portalwebAGPD/index-ides-idphp.php.

[3] Gobierno de España. Ley orgánica de protección de datos. URL https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750.

[4] Jawad Khalife. Novel approaches in traffic classification. 2016.

[5] Dr. Thomas Porter. The perils of deep packet inspection. URL https://www.symantec.com/connect/articles/perils-deep-packet-inspection.

[6] José Camacho, Pablo Padilla, F. Javier Salcedo-Campos, Pedro Garcıa-Teodoro, and Jesús Dıaz-Verdejo. Pair-wise similarity criteria for flows identification in p2p/non-p2p traffic classification. 2011.

[7] José Camacho, Pablo Padilla, Pedro García-Teodoro, and Jesús Díaz-Verdejo. A generalizable dynamic flow pairing method for traffic classification. 2013.

[8] Bro Team. Bro indice, . URL https://www.bro.org.

[9] Álvaro Maximino Linares Herrera. Repositorio del trabajo con bro. URL https://github.com/Lynares/bro-flows.

[10] Elvis Michael. La tarifa por hora de un programador. URL http://pyme.lavoztx.com/la-tarifa-por-hora-de-un-programador-12286.html.

[11] Jawad Khalife. A multilevel taxonomy and requirements for an optimal traffic-classification model. 2014.

[12] Internet Assigned Numbers Authority (IANA). Service name and transport protocol port number registry. URL https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[13] Christian Fuchs. Implications of deep packet inspection (dpi) internet surveillance for society. URL http://fuchs.uti.at/wp-content/uploads/DPI.pdf.

[14] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. URL http://ieeexplore.ieee.org/document/4738466/?reload=true.

[15] Kim Davies. Una introducción a iana. URL https://www.iana.org/about/presentations/davies-atlarge-iana101-paper-080929-es.pdf.

[16] Bro Team. Descarga de bro, . URL https://www.bro.org/download/index.html.

[17] Bro Team. Arquitectura de bro, . URL https://www.bro.org/sphinx/intro/index.html#architecture.

[18] Bro Team. Logs de bro, . URL https://www.bro.org/sphinx/script-reference/log-files.html.

[19] Bro Team. Framework netcontrol, . URL https://www.bro.org/sphinx/frameworks/netcontrol.html.

[20] Bro Team. Frameworks de bro, . URL https://www.bro.org/sphinx/frameworks/index.html.

[21] Bro Team. Trazas en bro, . URL https://www.bro.org/sphinx/components/trace-summary/README.html.

[22] securitykitten. Finding beacons with bro. URL https://gist.github.com/securitykitten/a7edcee0932c556d5e26.

[23] Bro Team. Tablas en bro, . URL https://www.bro.org/sphinx/script-reference/types.html#type-table.

[24] Bro Team. Tipo conn-id de bro, . URL https://www.bro.org/sphinx-git/scripts/base/init-bare.bro.html#type-conn_id.

[25] Bro Team. Tipo addr de bro, . URL https://www.bro.org/sphinx-git/script-reference/types.html#type-addr.

[26] Bro Team. Tipo port de bro, . URL https://www.bro.org/sphinx-git/script-reference/types.html#type-port.

[27] Bro Team. Tipo cennection, . URL https://www.bro.org/sphinx/scripts/base/init-bare.bro.html#type-connection.

[28] Bro Team. Tipo time, . URL https://www.bro.org/sphinx/script-reference/types.html?highlight=time#type-time.

[29] Bro Team. Tipo conn, . URL https://www.bro.org/sphinx/scripts/base/protocols/conn/main.bro.html#type-Conn::Info.

[30] Netresec. National cyberwatch mid-atlantic collegiate cyber defense competition (maccdc). URL http://www.netresec.com/?page=MACCDC.

[31] Bro Team. Instalación de bro, . URL https://www.bro.org/sphinx/install/index.html.

[32] Bro Team. Web de bro, . URL https://www.bro.org/sphinx/intro/index.html.

[33] Bro Team. Función get_port_transport_proto, . URL https://www.bro.org/sphinx/scripts/base/bif/bro.bif.bro.html#id-get_port_transport_proto.

[34] Bro Team. Analizadores de protocolos, . URL https://www.bro.org/sphinx/script-reference/proto-analyzers.html.

[35] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*. Pearson, 2010.

[36] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*, chapter 1. Pearson, 2010.

[37] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*, chapter 2. Pearson, 2010.

[38] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*, chapter 3. Pearson, 2010.

[39] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*, chapter 4. Pearson, 2010.

[40] James F. Kurose and Keith W. Rose. *Redes de computadores un enfoque descendente*, page 55. Pearson, 2010.