TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

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

Subtitulo del proyecto

Autor
Álvaro Maximino Linares Herrera

Directores
Jesús Esteban Díaz Verdejo

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 3 de septiembre de 2017

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

Subtitulo del proyecto

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

La identificación de tráfico en red es realmente importante para aplicaciones de ingeniería de tráfico y de seguridad.
En este trabajo se tratará la creación de un programa para un NMS (Network Monitoring System), en este caso se usará BRO, mediante el cual se pueda resolver el emparejamiento de flujos. BRO consiste en un NMS que funciona mediante el terminal en Linux o Mac, una de las peculiaridades de este programa es que para la creación de scripts que nos permitan extender la funcionalidad de la que dispone, tendremos que usar BRO como lenguaje de programación. Es un lenguaje de scripting, el cual está orientado a eventos, que se lanzan cuando ocurre algo relacionado con el control y análisis de redes, es un lenguaje que para los que vienen de C++, Java o Python, no debe de suponer un gran reto, más allá de acostumbrarse a sus sintaxis. Es un lenguaje potente que al estar orientado a redes nos permite obtener mucha información de los flujos que tenemos en la red o en el archivo que vayamos a analizar. En este trabajo mediante implementaciones offline se verificará la eficacia de esta técnica de clasificación de tráfico.

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 3 de septiembre de 2017.

D. **Jesús Esteban Díaz Verdejo**, Profesor del Área de XXXX del Departamento YYYY 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 3 de septiembre de 2017.

Los directores:

Jesús Esteban Díaz Verdejo **Nombre Apellido1 Apellido2 (tutor2)**

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].

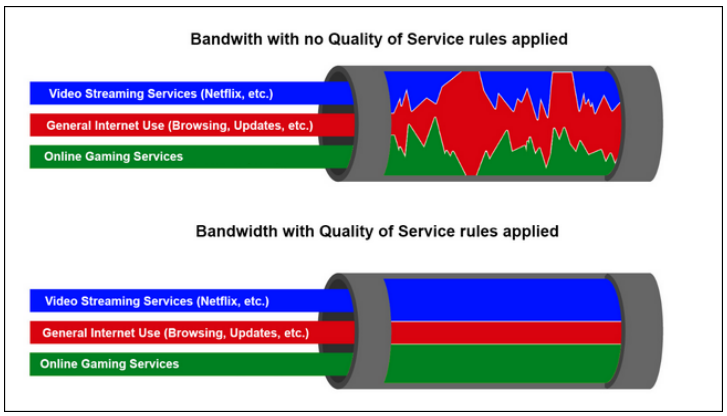

Figura 1.1: Ejemplo del ancho de banda sin calidad de servicio y con calidad de servicio.

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.

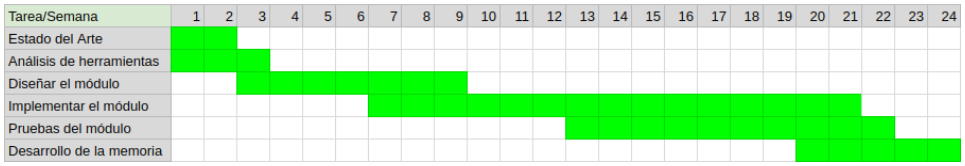

Figura 1.2: Temporización en diagrama de Gantt.

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 [11], 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 [12]. 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* [13].
- Por el contenido del paquete o *DPI* [14].
- Por la aplicación de técnicas de aprendizaje automático sobre estadísticas de tráfico, *machine learning* [15].

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 [16].

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 [15]. 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 [14]. 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 [10]. 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].

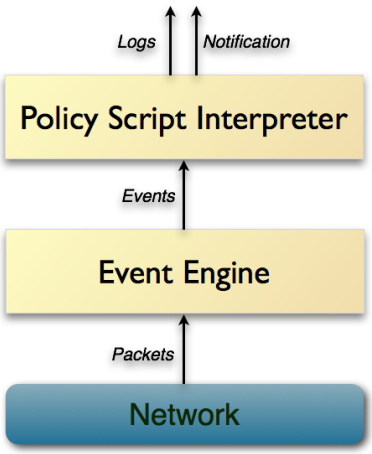

Figura 2.2: Arquitectura de Bro.

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.

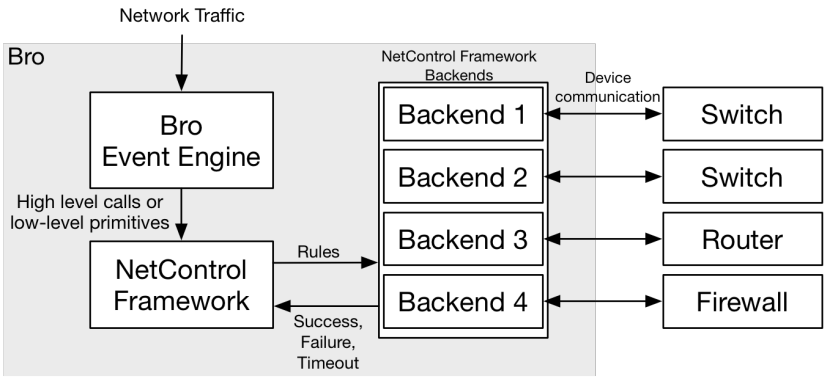

Figura 2.3: Arquitectura de NetControl.

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 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 (Apartado 2.3), 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 (necesaria).

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.

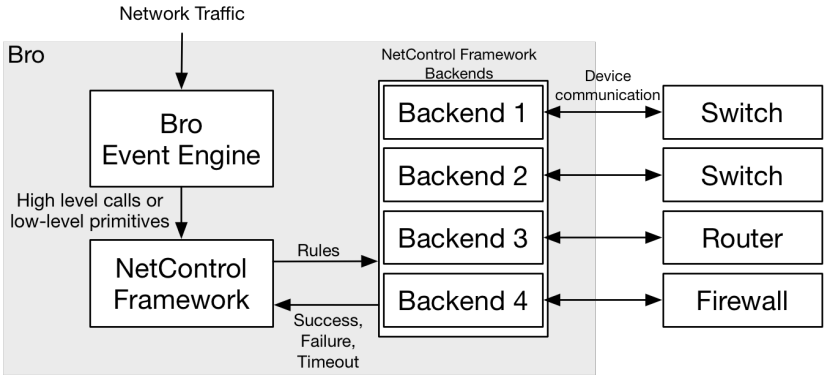

Figura 3.1: Arquitectura del módulo.

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, como se puede ver en la Figura 3.2, cuando un flujo es detectado tendrá el estado activo. En función de los distintos flujos que se vayan detectando los flujos activos se irán comparando con los nuevos que se detecten, de modo que los nuevos que 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.

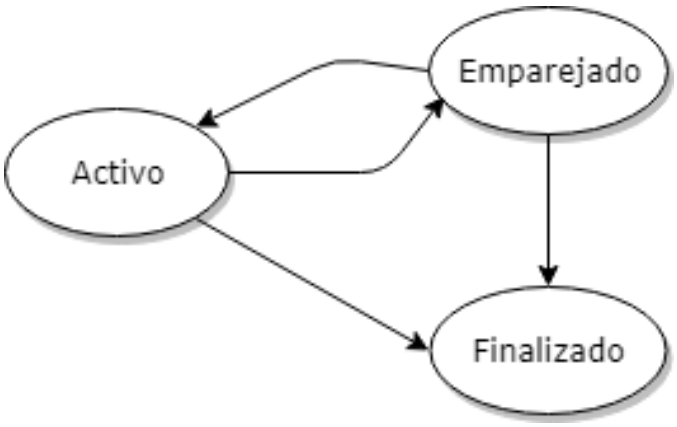

Figura 3.2: Distintos estados de los flujos.

Las entradas de tráfico para su análisis serán trazas en formato *pcap*. Las salidas serán registros, en los cuales se mostrarán los flujos que han sido emparejados. Ampliar!!!

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 pasa 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*.

Referenciar a la gestión de flujos

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, a los distintos flujos que se han detectado. 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.

- *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 aplicarán la función anterior.

Tras el uso de esta fórmula 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.

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. Esto se verá de forma más amplia en la siguiente sección.

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, serán mediante terminal. Esto puede suponer cierto inconveniente si se obtienen demasiadas salidas. También se pueden guardar las salidas en un fichero mediante el carácter *mayor que* . Con esto se guardará en un fichero en la ruta que se especifique, 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 flujos en Bro se realiza completamente con eventos. Por lo cual se tendrán que crear variables globales para el almacenamiento de los flujos que sean emparejables y los que estén activos.

El *nacimiento de un flujo* es controlado por un evento. Por lo tanto cuando se detecta un nuevo flujo se lanza un evento. Este evento se tendrá que controlar de forma que si se tiene ya un flujo activo con las mismas características, se compare y se almacene. Si por el contrario no se tiene ningún flujo con esas características se tendrá que almacenar directamente en el contenedor de flujos activos.

La *muerte de un flujo* también es controlada por un evento. Ahora lo importante es si es interesante a nivel del análisis seguir almacenando los flujos aunque estos hayan muerto. Si no se quiere tener almacenados flujos muertos se tendrá que eliminar de la estructura en la que está almacenado. Si se quiere seguir trabajando con ellos habrá que mantenerlos guardados en la estructura. Obviamente si el flujo que va a morir está emparejado con otro se borrará solo del contenedor de flujos activos. Manteniéndose en el contenedor de flujos emparejados. De lo contrario se perderá información. Todo esto habrá que decidirlo en el evento que gestiona la muerte del flujo.

Estos dos comportamientos de los flujos están controlados por eventos genéricos. Con un único evento se detecta que el flujo ha nacido y con otro evento si ha muerto, independientemente del tipo de protocolo al que pertenezca. No pasará esto con los distintos estados de los flujos que serán detectados. Pues no es lo mismo detectar un ACK de un flujo TCP que una respuesta de un flujo UDP. Serán tratados en eventos distintos y tendrán que ser gestionados con eventos distintos, cada uno destinado a un protocolo distinto.

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 podrá ver con mejor detalle en el apartado de implementación.

Dichas estructuras de almacenamiento, deberán de ser capaces de estar ordenadas por las IP's y los puertos. Lo cual se obtiene juntando las tablas con los vectores para así conseguir una especie de matriz bidimensional, la cual está 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 llega a almacenar muchos flujos.

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 si. Por lo tanto será de vital importancia comprenderlo para poder trabajar como se desea.

  Dentro del tipo de dato *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, siendo pues las IP's y los puertos.

  - *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 de puerto 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]

Para obtener más información sobre el tipo de dato *connection* lea [27].

- *time*.

  Este tipo de dato también es interesante. Aunque en otros lenguajes se puede obtener, en Bro es un tipo de dato por si mismo. Por lo tanto se podrá operar sobre él desde el principio, siendo una gran ventaja a la hora de calcular el tiempo de inicio de los flujos. Para leer más [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. Para leer más sobre esto [29].

Capítulo 4

Implementación

A continuación se contará cómo se ha implementado el módulo de Bro, resolviendo así el problema planteado.

La descripción del módulo será sin entrar en detalles de la programación. Se hará una descripción breve de los distintos eventos y funciones.

Lo primero que se va a detallar es la función para calcular el emparejamiento.

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

Esta función recibe como entrada dos flujos y devuelve un número de tipo *double*. En esta función lo que se hace es aplicar la fórmula de emparejamiento a los dos flujos que entran.

Al querer hacer la comparación lo 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 a tipo *count*, de esta forma se elimina la terminación con el tipo de protocolo del puerto. Se comentó anteriormente que se podía operar con esta terminación, pero al tener que operar con otros tipos de datos que no son puertos es mejor quitar la terminación, pues de lo contrario no se obtendrá un buen resultado.

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 más tipos de datos de carácter matemático.

El número de veces que se tiene un flujo con los mismos datos, o *Nip* en la fórmula, es de fácil cálculo. Gracias a la indexación basta con buscarlo en la tabla y calcular el tamaño del vector.

A continuación se procederá a la explicación de los distintos tipos de eventos usados y para que son usados.

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

Este evento recibe como entrada un flujo nuevo. Este flujo es una nueva conexión, la cual no está identificada previamente. Esto quiere decir o que es nueva o que ha sido borrada.

Este evento no devuelve nada, por lo tanto en el momento en el que es detectado el flujo se tendrá que crear un nuevo indice en la tabla de flujos activos o guardarlo para hacer la posterior comparación.

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

Ahora se verá cómo se van a gestionar la muerte de los flujos.

```
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 los demás una posición hacia atrás. De esta forma se seguirá teniendo un rendimiento óptimo.

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

```
event connection_established(c: connection)

event connection_finished(c: connection)

event udp_request(u: connection)

event udp_reply(u: connection)
```

Los dos primeros eventos son los correspondientes al tráfico *TCP*. Los otros dos, como se puede ver 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, son los únicos eventos que detectan conexiones de tipo *UDP*.

Dentro de todos estos eventos se realiza lo mismo. Primero se comprueba que los dos flujos no son el mismo, si son el mismo se termina el análisis. Si no son lo mismo se pasa a la función de emparejamiento ya descrita y se compara el número que se obtiene con el umbral que se ha definido antes de la ejecución. Si el resultado es mayor que el umbral son emparejables, por lo cual se tendrá que guardar en la tabla de emparejados y se informa mediante un mensaje en pantalla de que lo son. Si es menor que el umbral no son emparejables y mediante un mensaje en pantalla se informa de que no son emparejables.

Es necesario recordar que se puede dar que se puede estar usando dos eventos o más a la vez, por lo que los mensajes en pantalla pueden ser confusos. Se tiene que tener en cuenta que hay cierto retardo en los mensajes, por lo que lo interesante es el resultado final de las tablas.

Además de estos eventos se tienen los siguientes eventos genéricos.

```
1 event bro_init()
2
3 event bro_done()
```

El primero se lanza cuando Bro se inicia y mostrará el tiempo de inicio. El segundo se lanza cuando Bro finaliza, por lo tanto es el último evento que se lanzará y mostrará la hora de finalización. Con estos dos eventos se tendrá control de cuando fue lanzado el análisis y cuando finalizó.

Aparte de estos eventos, también existen dentro del módulo otros dos eventos destinados a detectar paquetes del protocolo *ICMP*. Este tipo de eventos son necesarios pues Bro los detecta igual que los de tipo *TCP y UDP*.

```
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 si un router o un host no son alcanzables.

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.

Aunque estos dos eventos tienen más parámetros de entrada que los eventos anteriores, para calcular el emparejamiento solo será usado el primer parámetro, el cual hace referencia al flujo. El trato del flujo dentro de los eventos es el mismo que el que se aplica a los eventos *TCP y UDP* anteriormente descrito.

A continuación se verá cómo se ha implementado las estructuras de almacenamiento de los flujos.

```
1 global collection: table[addr, addr, port, port] of vector of
      connection &synchronized;
2
3 global collection_added: table[addr, addr, port, port] of vector of
      connection;
```

Se trata de dos tablas globales, cuyo indice está constituido por las IP's de origen y destino y los puertos de origen y destino. Además como los indices son únicos, cada índice apunta a un vector, y dentro de dicho vector se almacenan los flujos ordenados dependiendo de cuando son detectados.

La primera tabla almacena los flujos que están activos. La segunda almacena los que ya están emparejados.

Capítulo 5

Evaluación y pruebas

Capítulo 6

Conclusiones y trabajo futuro

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

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

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

[13] 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.

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

[15] 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.

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

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

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

[32] 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.

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

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

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

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

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

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

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