
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, 3 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

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

La clasificación de tráfico en red es una tarea importante en lo relativo a las comunicaciones, en un mundo cada vez más digitalizado e intercomunicado, lo que más importa es la seguridad y para ello es esencial la clasificación del tráfico, permitiendo detectar de forma temprana intrusiones y comportamientos anormales, para los cuales podremos prepararnos, de esta forma, por ejemplo, el encargado de un servidor podrá mantener la calidad de servicio, pues mediante ISP (Internet Service Provider) se puede establecer diferentes niveles de prioridad en el tráfico de red.

En este trabajo haremos uso de BRO, un NMS (Network Monitoring System), al cual mediante la implementación de técnicas de emparejamiento de flujos, lo dotaremos de la capacidad analítica de discernir que flujos son emparejables, y por lo tanto pertenecen a lo mismo.

## 1.1. Conceptos básicos

En este apartado vamos a aclarar los conceptos básicos que son necesarios para entender este trabajo.

Lo primero que tenemos que tener claro es lo que es un paquete. Un paquete es cada un fragmento de la información que queremos enviar a través de una red. Este paquete contiene datos sobre quiénes somos, digitalmente, y hacia donde enviamos la información, así que incluso un sólo paquete da mucha información sobre quienes somos en la red.

Lo siguiente que debemos de saber es que es un flujo, para ello tenemos que volver hacia la definición anterior, pues consiste en la historia de un grupo de paquetes, esto quiere decir cómo se mueven los paquetes a través de las capas de red, de datos y la física. Para quien no sepa sobre redes se estará preguntando que son estas capas, pues bien, estas capas son del mo-

delo OSI, que es el modelo de referencia para los protocolos de red, siendo las distintas capas las siguientes:

1. Aplicación 2. Presentación 3. Sesión 4. Transporte 5. Red 6. Enlace de datos 7. Física

Lo siguiente que explicaremos será que es un NMS o Network Monitoring System, como el nombre nos indica consiste en un programa que se dedica a monitorizar el tráfico del sistema y en caso de encontrar algún problema avisará al administrador del sistema. En nuestro caso el NMS que usaremos será Bro, aunque existen otros.

## 1.2. TCP y UDP

Describir que es TCP y que UDP, poner imágenes del libro sobre la comunicación.

# Capítulo 2

# Estado del arte

La idea de la que partimos es la siguiente: Los flujos son un conjunto de paquetes que comparten la misma información, IP de origen, IP de destino y los puertos de origen y destino, aparte de esta información también tienen un tiempo de inicio, relativo al timestamp del primer paquete del flujo y una duración. Con estos datos ya podemos empezar a trabajar en el emparejamiento, pues si dos flujos comparten las IP's de origen y destino y los puertos de origen y destino ya serían candidatos a la comparación. Para comparar los flujos lo que haremos será aplicar la siguiente fórmula:

Donde tenemos que G(x,y) es:

Tenemos que x e y son los flujos a comparar, NIP(x,y) es el número de veces que tenemos almacenado flujos similares, por lo cual siempre será uno, como mínimo, dp1(x,y) son los puertos de origen de los dos flujos, que en este caso serán los mismos dp2(x,y) son los puertos de destino, k1 y k2 son constantes predefinidas antes de ejecutar el programa, estando estas normalmente en los trabajos anteriores definidas entre 1 y 10000 y, por último, dt es la diferencia de tiempo entre los timestamps de los dos flujos.

Con esta fórmula obtendremos un número que podremos comparar con un umbral que estará predefinido, si el umbral definido es pequeño tendremos más flujos almacenados, si el umbral es más grande serán menos los flujos que serán emparejables y por tanto almacenados. [1]

## 2.1. Capas de Red

En nuestro caso nos quedaremos solamente en la capa de transporte, despreciando pues la información de los flujos en cuanto a la capa de aplicación, HTTP, SMTP, aplicaciones P2P y demás... y usaremos un poco de la capa de red, como ICMP, pero solo para añadir más información a los distintos flujos que podemos llegar a analizar, pero despreciaremos la información relativa al resto de esta capa, como puede ser el enrutamiento interno de un sistema autónomo de Internet: RIP, de un AS en INternet: OSPF, entre sistemas autónomos BGP. Más que nada porque Bro no nos permite acceder a esta información, por lo tanto es imposible en este trabajo hablar sobre ella, aunque si nos permite analizar cosas de la capa de aplicación, aunque no siempre, pues es información opcional y por lo tanto puede estar o no. En el caso del archivo pcap que utilizamos para la prueba, nitroba.pcap, no está disponible esta información. Esta información se puede consultar al mostrar toda la información del flujo.

## 2.2. DPI

La Inspección Profunda de Paquetes o DPI por sus siglas en inglés, Deep Packet Inspection, consiste en analizar un paquete entero, no solo la cabecera del paquete. Gracias a esta técnica podemos detectar intrusiones, ataques de denegación de servicio y demás [3], lo cual es bastante interesante, ya que podremos saber qué medidas tomar respecto a los paquetes maliciosos. Este análisis también sirve para realizar estadísticas sobre el tráfico de una red, que en nuestro caso nos interesa bastante, pues nosotros vamos a realizar emparejamiento de flujos, mediante la información de sus paquetes, siendo el tiempo del primer paquete una de las medidas que usamos como variable para la comparación de los flujos. Esta técnica es más costosa que la inspección de paquetes poco profunda, la cual consiste en analizar solamente las cabeceras de los paquetes, por lo cual es más rápida y menos costosa, y las dos tratan de asegurar la neutralidad de la red.

## 2.3. NMS y BRO

Bro [2] es un NMS, Network Monitoring System, cuya principal actividad es analizar el tráfico de una red para la búsqueda de amenazas. Una de las muchas peculiaridades de Bro es que usa un lenguaje propio, llamado también Bro, el cual tiene similitudes con Python y Java, y se gestiona mediante eventos. En el caso de este trabajo se utilizará dicho lenguaje, el cual en un principio puede parecer dificil, pero que acaba siendo bastante útil para la tarea que queremos, pues tiene un tipo de dato llamado connection, el cual está relacionado con los flujos, siendo al final este tipo el

propio flujo. En este tipo se establecen bastantes datos que nos serán útiles, como la IP de origen, la de destino, así como los puertos y sus protocolos, también guarda la información de la duración así como el timestamp y más información que en este caso no será relevante, como la historia del flujo, donde se nos informa del tipo de paquetes que contiene e incluso el número de paquetes.

En el caso de los eventos, también nos facilitan el trabajo, pues tenemos eventos que gestionan todo lo que tiene que ver con los flujos, desde cuando aparece un nuevo flujo el cual no tenemos identificado previamente, lo cual para nuestro trabajo es fundamental, pues nos hace el programa más eficiente, ya que cada vez que se lanza ese evento significa que tenemos que añadir ese flujo a la lista de flujos comparables, sin tener que compararlo con los que tengamos previamente almacenados. También tenemos flujos que nos indicarán cuando un flujo esté a punto de morir, siendo el momento de eliminarlo de la lista de comparables y, en caso de tener flujos con los que éste ha sido comparado, pasará el primero de estos a reemplazar el flujo que va a morir en la lista de comparables, de ésta forma nos evitamos tener que estar comprobando la duración del flujo para eliminarlo manualmente. Con todo ésto conseguimos un programa eficiente en cuanto a tiempo y que, gracias a estos eventos, va a identificar tres protocolos distintos, siendo estos TCP, UDP e ICMP. Cabe destacar que los eventos de UDP empleados en nuestro programa son muy costoso, por lo tanto en caso de tener mucho tráfico UDP, siendo de dos tipos distinto "reply" y "request", puede hacer que el análisis tarde más tiempo del deseado, pero el programa suele funcionar en una horquilla de tiempo bastante aceptable en términos de rendimiento para el análisis del tráfico. En el apartado de "resolución del trabajo" se hablará sobre los eventos usados con más detalle.

Bro también nos otorga una gran cantidad de logs en los cuales él mismo puede llegar a clasificar el tráfico dependiendo del protocolo que se use, siendo algunos ejemplos, HTTP, DNS, SSL, etc., aunque como en nuestro caso usamos un archivo pcap esta información dependerá de si han sido cargados los scripts correspondientes cuando se realizó la captura del tráfico, pues en los flujos de Bro la información relativa a HTTP, DNS y demás es opcional, siendo únicamente obligatorio, en lo relativo a puertos que se informe de si pertenece a TCP, UDP o ICMP.

Bro trabaja sobre Linux y sobre la terminal por lo tanto será necesario leer la documentación para saber que opciones nos interesa tener activadas cuando vayamos a analizar el tráfico, pues al no tener interfaz gráfica nos puede ocurrir que no tengamos activado lo que queramos y otras cosas que no precisamos si estén activas.

También es digno de mención que la información que hay en la red sobre Bro es bastante escasa, quedando casi exclusivamente la información que se puede encontrar en la web de Bro[4].

### 2.3.1. Requisitos de Bro

Podemos descargar los binarios y el código fuente desde la web de Bro [5], pero en mi caso después de experimentar ciertos fallos con ésta instalación, recomiendo seguir la instalación de Bro de su web en el apartado de instalación [6], pues allí nos guía a una instalación que es más fácil de mantener actualizada, ya que descargamos el código fuente directamente del repositorio de GitHub, por lo cual si hay algún cambio importante, con alguna mejora interesante la podremos tener casi directamente a nuestra disposición.

Para la instalación con los archivos de GitHub, tendremos que hacer lo siguiente:

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

Estos son los prerrquisitos para la instalación de Bro, ahora clonamos el repositorio de GitHub en nuestro ordenador:

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

Y cuando termine de descargarlo nos metemos en la carpeta y ejecutamos lo siguiente:

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

De esta forma ya tendremos instalado Bro en nuestro ordenador y podremos empezar a usarlo.

Ahora para usarlo tendremos que hacer lo siguiente, lo primero será ajustar el PATH de nuestro terminal, para ello, tendremos que ejecutar en nuestro terminal lo siguiente:

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

Es importante hacerlo, pues sino nos saltará un error diciendo que Bro no se encuentra instalado. Una vez hecho esto, podemos empezar realizando un script sencillo y probar a ejecutarlo, un ejemplo de script sencillo es el que se ve en el apartado "Ejemplo evento que se lanza cuando se crea un nuevo flujo", supongamos que el script se llama prueba.bro, pues para lanzarlo tendremos que hacer lo siguiente:

```
~$ bro -b -r pcap/nitroba.pcap scripts/bro-flows/prueba.bro
```

En este caso dentro de la carpeta Bro que se genera cuando clonamos el repositorio he creado una carpeta llamada scripts, para tener todos los scripts allí guardados, por lo tanto al encontrarnos en la carpeta raíz de Bro, tendremos que ponerlo en la ruta, la opción -b que pongo en la orden es solo para este ejemplo, lo que hace es no cargar los scripts que se encuentran en el directorio base/. Como necesitamos leer de un archivo pcap, tenemos que poner la opción -r, que indica a Bro que tiene un archivo de datos que leer, al igual que con los scripts, también hemos creado una carpeta pcap en la cual guardamos los archivos pcap que queramos leer. Una vez que lo ejecutemos se nos mostrará en el terminal un mensaje cada vez que se genere un flujo nuevo, cuando se exceda el timeout también y cuando un flujo vaya a ser eliminado de la memoria.

### 2.3.2. Gestión de tráfico en Bro

#### Nacimiento y muerte de flujos

En Bro el nacimiento de un flujo es controlado por un evento, dicho de otro modo, cuando aparece un nuevo flujo se lanza un evento, de ésta forma podremos controlar cuando un flujo nuevo aparece y podremos analizar toda su información en este evento o en los siguientes antes de que el flujo muera, o en el caso de tenerlo almacenado, antes de eliminarlo, pues una vez que lo eliminemos no podremos recuperarlo, por lo tanto para el análisis detallado del tráfico es interesante guardarlo.

La muerte funciona también por eventos, siendo activado el evento cuando el flujo está a punto de morir, por lo tanto todavía podremos operar sobre él, por ejemplo, podremos decidir si queremos que el flujo sea eliminado o seguir guardándolo para análisis estadísticos.

Por lo tanto como se comentó en apartados anteriores en Bro la mayor parte de la información se gestiona con eventos, lo cual facilita enormemente el trabajo que tenemos por delante.

### Marcas temporales

Es interesante remarcar el uso de los timestamps, pues como ya se sabe son una variable importante que usaremos para la comparación necesaria para emparejar los flujos, cierto es que los datos más importantes son las IP's y los puertos, pues si no son iguales no procederemos a aplicar la fórmula necesaria para el emparejamiento.

En Bro, como lenguaje de programación, volvemos a encontrarnos con un tipo de dato propio para la gestión de estos datos, en este caso el tipo es time [5], cuyo nombre es de por si muy indicativo de lo que va a contener. La peculiaridad de este tipo de dato es que el tiempo viene dado en valor absoluto, por lo cual tendremos que formatearlo para representarlo, aunque no será necesario si queremos operar con él. Bro dispone de funciones para pasar de double a time, double_to_time, obtener el tiempo actual, current_time, y por último también disponemos una función para obtener el tiempo del último paquete procesado, network time, obteniendo así su timestamp, aunque parezca que necesita estar procesando de manera online para usar esta función no es así, pues también sirve para los archivos de datos de tráfico de red guardados, como los archivos pcap que usaremos nosotros para el emparejamiento. Para mostrar correctamente el valor del tiempo podemos usar strftime, lo cual nos formateará el tiempo y lo mostrará acorde a lo que le digamos en la cabecera de la función. También contamos con el caso contrario, es decir, pasar de string a time, para ello tendremos que usar la función strptime.

Ahora una de las funciones más importantes es time_to_double, que sirve para pasar un tipo time a double, haciendo más fácil operar con él.

### Listado de flujos activos

Para mantener un listado de flujos activos haremos uso de un contenedor para almacenarlos y usaremos de los eventos de Bro para gestionarlos, primero cuando se generen flujos que previamente no habíamos visto los guardaremos para compararlos, si aparecen flujos que ya tenemos habíamos visto, los compararemos con los almacenados y si nos interesa los guardaremos, y por último cuando Bro nos indique que algún flujo va a ser eliminado de la memoria y por lo tanto ese flujo ya no está activo, veremos si hay algún candidato a ocupar su puesto en el listado de flujos emparejados y lo eliminamos del listado de flujos comparables.

Para mantener un listado de flujos activos Bro nos proporciona varias formas. La primera es usar un vector, el cual es fácil de usar si venimos de lenguajes como C++, podremos incluir archivos usando los corchetes, [ ],

indicando el índice dentro de ellos, para borrar un elemento usaremos delete y para consultar el tamaño del vector usaremos —v—, pero sin embargo por si sólo es muy ineficiente para almacenar datos del tipo connection, pues solo dispone de índices numéricos.

Otro tipo para almacenar datos es set, el cual para borrar y consultar el tamaño es igual que para los vectores, sin embargo, para agregar datos tendremos que usar add se[s], estando entre los corchetes lo que queremos almacenar, aunque también podemos almacenar datos redifiniendo el set, por ejemplo, s1 = set(1,2,3). El set también es ineficiente, aunque menos, pues en un índice, aunque podemos guardar varios elementos estos tienen que ser únicos y corremos el riesgo de eliminar los datos almacenados.

Por último podemos usar un tipo table, que es básicamente un map, de los que se usan en C++, pudiendo hacerlos bastante interesantes, pues podemos hacer un table de connection de vector o de set de connection, por lo tanto esta será la forma que usaremos para almacenar el listado de flujos activos, pues nos da bastante versatilidad a la hora de manejar el listado de los flujos, porque podemos acceder por índice a la lista de todos los flujos que comparten datos, haciendo que sea una tarea bastante asequible, en términos de eficiencia.

### 2.3.3. Identificación de flujos en Bro (Tipo de flujo/Protocolo)

Como ya se explicó en otro apartado, Bro consta del tipo connection [6] para referirse y tratar los flujos, por lo tanto como en el flujo hay información sobre el protocolo que usa podemos consultarlo haciendo referencia al flujo que estamos tratando. Hay que destacar que dentro de connection, en history, existe otro tipo llamado Conn::Info [7] que extiende aún más los datos de connection, y siendo capaz sus datos de crear los logs de lo que estemos analizando, por lo que podremos generar un log con los protocolos que son usados en la sesión. Cabe destacar que mostrando la información del puerto mediante el tipo port, el string que nos muestra lleva el número del puerto y el protocolo al que pertenece, pudiendo así extraer si es TCP, UDP o ICMP. Aunque si lo que queremos es obtener solamente el protocolo de transporte al que pertenece el flujo podemos hacer uso de la función get_port_transport_proto [8].

Es destacable que el tipo de dato port soporta la comparación, lo cual será usado en el programa.

También en las múltiples extensiones de Bro encontramos un analizador de protocolos, protocol analizer [9], el cual dispone de varias funciones in-

teresantes para el análisis de protocolos.

### 2.3.4. Ejemplo de uso de eventos

Ahora que tenemos algunos conceptos claros pondremos el ejemplo de un programa en el cual se lanza un evento cada vez que aparece un flujo nuevo.

```
## Para todo tipo de conexiones, ya sean TCP, UDP o ICMP. Este evento salta con cada nue
## conexion, con el primer paquete de una conexion desconocida, por lo que mirando el
## documento generado vemos que esto es un nuevo flujo.
event new_connection(c: connection)
  {

   if ( connection_exists( c$id ) ) {
      print fmt("Nueva conexion establecida Timestamp: %s desde %s a %s", strftime("%Y/%
      print fmt("Protocolo del puerto: %s", get_port_transport_proto(c$id$orig_p));
      print fmt("Informacion de las 4 tuplas del paquete: %s", c$id);
   } else {
     print fmt("La conexion ya existe");
     print fmt("_____________________________________________________________");
   }

 }

## Solo disponible para conexiones TCP, se genera cuando no hay actividad en un
## periodo de tiempo determinado.
event connection_timeout(c: connection)
  {
    print fmt("Conexion TCP ha excedido el timeout: %s",c$id);
  }
## Este evento salta para todo tipo de conexion, se da cuando el estado interno
## esta a punto de eliminarse de memoria
event connection_state_remove(c: connection)
{

  print fmt("Conexion de %s a %s a punto de ser eliminada de la memoria", c$id$orig_h, c

}
```

Algunos comentarios sobre este código. Como habrá podido notar en el lenguaje de Bro los comentarios se realizan con dos almohadillas, ##, los print son como en la gran mayoría de lenguajes, con la particularidad de que si vamos a mostrar un string debemos de indicar que va formateado mediante fmt, %s es para mostrar el string, tal y como se hace en C con los datos de tipo int mostrándolo con %i, por ejemplo. Por último para acceder a la distinta información que contiene el tipo de dato debemos de usar $, esto es parecido a como accedemos en Java a los datos de las clases.

Aunque el código está comentado merece la pena hacer algunas valoraciones sobre los eventos usados, el evento new_connection es muy útil, pues se ejecuta cada vez que llega un flujo que no ha sido usado antes y además re-

conoce TCP, UDP e ICMP. El evento connection_timeout aunque sólo sirva para flujos del protocolo TCP es bastante útil para empezar a trabajar sobre Bro, pues sirve para empezar a comprender como funcionan los eventos. Por último el evento connection_state_removed se lanza para TCP, UDP e ICMP cuando los flujos están a punto de morir, por lo tanto tiene un gran valor para eliminar de memoria los flujos que ya no nos interesan porque no están activos, y si este evento no se lanza no serán eliminados, en caso de que los flujos estén guardados.

# Capítulo 3

# Objetivos

En este trabajo lo que se trata de hacer es:

- Demostrar que es posible analizar el tráfico mediante emparejamiento de flujo.
- Realizar un programa que sea capaz, a partir de un archivo pcap, analizar los flujos.
- Que dicho programa muestre la información de los flujos emparejado.
- Implementación de la función de la comparación que se halla en el ensayo [1].

Para demostrar que es posible analizar el tráfico mediante emparejamiento de flujo, y por tanto con DPI, se usará Bro para gestionar el tráfico de la red, aunque al ser una implementación offline lo que se pretende realizar tendrá que leer datos de un archivo pcap.

Al realizar una extensión para Bro tendremos que adaptarnos y aprender a programar en el lenguaje que utilizan para ello, Bro, el cual tiene ciertas peculiaridades, aunque como ya sabemos programar en distintos lenguajes, el aprendizaje de este lenguaje es muy rápido, pues la gran mayoría de las cosas ya sabemos hacerlo, sólo falta adaptarse a las peculiaridades que tiene, como cualquier lenguaje. Alguna de estas peculiaridades es por ejemplo el acceso a los datos dentro de un tipo, que se realiza mediante $, cuando en otros lenguajes accedemos a ellos mediante . ó -¿.

Para mostrar la información de los flujos será necesario crear una función que nos muestre por pantalla que contienen dichos flujos. Realizar la función es para no repetir código, lo cual entra dentro de las buenas prácticas de programación. En dicha función se accederemos a los datos de IP origen y destino y los puertos origen y destino de los flujos que estamos comparando,

esta función no realiza comparaciones, simplemente muestra la información, pues esta función es llamada después de encontrar dos flujos que son emparejables.

Para ver si dos flujos son emparejables, primero tendremos que ver si las IP's y los puertos de origen y destino son iguales. En caso de serlo pasaremos a aplicar la función, que se encuentra en el primer apartado, la cual nos devolverá un número que compararemos con un umbral que nosotros definamos, haciendo que si es mayor no sean emparejables y si es menor que dicho umbral si sean emparejables.

# Capítulo 4

# Implementación

Ahora vamos a proceder a explicar cómo se ha solucionado los problemas propuestos, siendo estos explicados a través del programa que se ha realizado, quedando así más clara la explicación.

```
## funcion para la comparacion de los flujos, c1 el flujo que esta en el set conex y c2
function emparejamiento(c1: connection, c2: connection ):double {

  local Nip=1; ## Variable para saber cuantas conexiones tenemos
  local Po1: count; ## Puerto origen del primer flujo
  local Po2: count; ## Puerto origen del segundo flujo
  local Pd1: count; ## Puerto destino del primer flujo
  local Pd2: count; ## Puerto destino del segundo flujo
  local k1 = 1;  ## Variable fija
  local k2 = 10; ## Variable fija
  local dt: double; ## Variable para la diferencia de los tiempos
  local resultado = 0.0; ## Lo ponemos a 0
  print c1$uid;
  print c2$uid;
## Podemos saltarnos este bucle si inicializamos Nip a 1
  ## for (s in conex){

  ##   if((s$id$orig_h == c1$id$orig_h) && (s$id$resp_h == c1$id$resp_h) && (s$id$orig_p
  ##           Nip=Nip+1;
  ##           print fmt("Numero de Nip sin table: %d", Nip);
  ##           break;
  ##   }
  ## }

  if(c1$uid==c2$uid){
```

```
    print fmt("Son el mismo flujo, no se realiza incremento en Nip");
  }else{
## Este bucle lo puedo hacer sin ningun problema, pues en los eventos todavia n
  for (i in empa){
    if((i$id$orig_h == c2$id$orig_h) && (i$id$resp_h == c2$id$resp_h) && (i$id$
            Nip=Nip+1;

    }
  }
  print fmt("Numero de Nip en table: %d", Nip);
  informacion_coincidencia(c1,c2);
  print fmt("Tiempo de inicio del flujo: %s", |c1$start_time|);
  print fmt("Tiempo de inicio del flujo: %s", |c2$start_time|);
  ## Para dp1 y dp2 que son 1-norm usamos la "Manhattan norm" que dice lo sigui
  ## k1 y k2 son dos variables que nosotros le ponemos de forma manual, en este
  ## dt es la diferencia de tiempo entre los time stamp de los primeros flujos
  ## el tipo time se supone que es como un double, por lo tanto podremos restar
  ## para la comparacion de puertos primero tendremos que hacer uso de la funci
  ## la cual nos pasa el puerto, que recordamos que va tambien con un string en
  ## valor numerico que si podremos restar sin problemas
  ## La funcion quedaria asi: (Nip-1)+(1/(dp1+k1))+(1/(dp2+k1))+(1/(dt+k2))
  Po1=port_to_count(c1$id$orig_p);
  Pd1=port_to_count(c1$id$resp_p);
  Po2=port_to_count(c2$id$orig_p);
  Pd2=port_to_count(c2$id$resp_p);
  ## local t1: double;
  ## local t2: double;
  ## t1 = time_to_double(c1$start_time);
  ## t2 = time_to_double(c2$start_time);

  dt=(|c1$start_time| - |c2$start_time|);

  ## print fmt("Tiempo paquete 1: %s", t1);
  ## print fmt("Tiempo paquete 2: %s", t2);
  print fmt("Diferencia de tiempo: %s", dt);
  resultado=(Nip-1)+(1/((Po1-Po2)+k1))+(1/((Pd1-Pd2)+k1))+(1/(dt+k2));
 }
 return resultado;

}
```

En este trozo de código se muestra la función que se usa para devolver el resultado de la función de comparación. El resultado será comparado con el umbral que nosotros definimos, haciendo que si dicho resultado es ma-

yor que el umbral serán emparejables y por tanto serán guardados en un table[connection] of connection, lo que sería comparable en C++ con un map. Decidimos guardarlos dentro de cada uno de los eventos que lanzamos cuando Bro reconoce un determinado protocolo, dichos eventos son:

- connection_established, el cual se lanza cuando Bro localiza un SYN-ACK que corresponda a un handshake de TCP.
- connection_finished, el cual lanza Bro cuando una conexión TCP finaliza de forma normal.
- udp_request, el cual se lanza cuando Bro localiza un flujo que corresponde con UDP lanzado desde el origen.
- udp_reply, que es lanzado por Bro cuando localiza una respuesta a un flujo del anterior evento.
- icmp_echo_request, lanzado por Bro cuando localiza un flujo de tipo ICMP de echo.
- icmp_echo_reply, el cual es lanzado por Bro cuando localiza una respuesta a los flujos del evento anterior.

Dentro de estos eventos descritos se realiza la comprobación de que dos flujos se pueden comparar, ya que tienen los mismos IP's y puertos de origen y destino. Una vez en la función de comparación se almacenarán dichos puertos, pasándolos al tipo count para su uso en la función, se establecerán dos constantes, se calculará la diferencia de tiempo y en una variable se almacenará el número de flujos que hay con esos datos. Cuando realicemos esto, aplicaremos la función de cálculo y podremos decir si los flujos son emparejables.

Para mostrar la información de los flujos tenemos dos opciones, de las cuales mostramos el código a continuación:

```
## Creo funcion auxiliar para ver la informacion del flujo nuevo que se añade, no de toc
function informacion_flujo(c: connection){
    print fmt("Informacion del flujo nuevo IPo: %s , Po: %s , IPd: %s , Pd: %s ", c$id$o
}


## Creo funcion auxiliar para ver la informacion del flujo que se coincide
function informacion_coincidencia(c: connection, p: connection){
```

```
    print fmt("Informacion del primer flujo  IPo: %s , Po: %s , IPd: %s , Pd: %
    print fmt("Informacion del flujo coincidente  IPo: %s , Po: %s , IPd: %s , 
}

La función informacion_flujo sirve para mostrar la información del flujo relati

En el programa se hace uso también de un evento para eliminar los flujos que es

connection_state_removed, evento que lanza Bro cuando va a eliminar de la memor

## Cuando la conexion va a ser borrada la eliminamos del set y en caso de tener
## se obtienen los mismos flujos añadidos que eliminados, por lo tanto hay que 
## Sirve para TCP, UDP e ICMP
## Generated when a connection's internal state is about to be removed from mem
## once for every connection when it is about to delete the internal state. As 
## script-level cleanup that needs to be performed for every connection.
## This event is generated not only for TCP sessions but also for UDP and ICMP 
event connection_state_remove(c: connection){

## Creo un connection local para poder pasarlo de empa a conex
   local cl: connection;
## Variable booleana para controlar el acceso al set
   local esta = F;

## for que va recorriendo el set y haciendo comparaciones
    for(s in empa){
      if((s$id$orig_h == c$id$orig_h) && (s$id$resp_h == c$id$resp_h) && (s$id$
## Si se dan todas las condiciones la variable booleana de control de acceso al
              esta=T;
## Al existir otro flujo lo copiamos en cl
              cl=s;
              break;
      }
    }

    ## Aqui si tenemos otro flujo igual al que vamos a eliminar lo metemos en c
    ## Con la variable booleana controlamos el decrecimiento del set
    if (esta==T){
      delete conex[c];
      add conex[cl];
      delete empa[cl];
      ## print fmt("Hemos borrado");
      ## print empa[cl];
    } else {
```

```
      delete conex[c];
    }
    elimi=elimi+1;
    ## Quitamos uno al tamaño del set
    tams=tams-1;
    esta=F;
    ##  print fmt("Tamanio del set: %d", tams);
    ##  informacion_flujo(c);

}
```

En nuestro caso antes de eliminarlo del set de comparaciones lo que hacemos es ver si en la tabla de emparejados existe un flujo con los mismos datos, en caso de que exista borramos el flujo del set, pasamos al set el flujo de la tabla de emparejados y este flujo lo eliminamos de la tabla. La ventaja que tiene esto es que el timestamp será actualizado, haciendo que la diferencia de los tiempos de dos flujos no sea demasiado grande y por lo tanto teniendo más posibilidades de que exista algún flujo que sea emparejable con el nuevo. En caso de que no exista un flujo con las características deseadas para la sustitución en el set de conexiones, simplemente será borrado, si no ha tenido ningún flujo con el cual ha sido emparejado en todo el uso del programa, este flujo borrado nunca será mostrado, en caso de que si lo tuviese, el flujo siempre estará almacenado en una tabla que mostramos al finalizar el programa.

Para añadir flujos al set de conexiones, tenemos un evento dedicado a ello:

- new_connection, Bro lanza este evento cada vez que detecta un flujo que era desconocido.

```
## Cada vez que entra un nuevo flujo lo comparo con lo que ya tengo en el set
## Este evento se lanza con cada nueva conexion de un flujo que no sea conocido
## Generated for every new connection. This event is raised with the first packet of a p
event new_connection(c: connection){

## Si el set esta vacio meto el primer flujo
   if(tams==0){
    add conex[c];
   }
## Sumamos uno al tamaño del set
    tam=tam+1;

## Variable booleana para controlar el acceso al set
```

```
     local met = F;

## for que va recorriendo el set y haciendo comparaciones
     for(s in conex){
## Copiamos en la variable local para comparar con todo lo que hay en el set
       if((s$id$orig_h != c$id$orig_h) && (s$id$resp_h != c$id$resp_h) && (s$id$
               ## Si se dan todas las condiciones la variable booleana de contr
               met=T;
       }

     }
    ## Con la variable booleana controlamos el crecimiento del set
     if (met==T){
       add conex[c];
       tams=tams+1;
       ## print fmt("Meto un flujo nuevo por la conexion de origen distinta");
     }
     met=F;
    ## print fmt("Numero de flujos al momento: %d", tam);
    ## print fmt("Tamanio del set: %d", tams);
    ## informacion_flujo(c);

}
```

Lo primero que debemos de hacer es ver si el tamaño del set de conexiones es 0, en caso de serlo añadimos el flujo directamente, en caso de que no sea 0 tenemos que ver primero si tenemos un flujo con los datos de comparación iguales, en caso de tenerlo el flujo no será añadido, en caso de que no exista otro flujo con los mismos datos de comparación lo guardaremos.

Por último usamos dos eventos en el programa para ver cuando se lanza y cuando finaliza el programa:

- bro_init, evento que se lanza cuando iniciamos Bro.
- bro_done, evento que se lanza cuando Bro va a terminar.

```
## Evento que se lanza cuando se inicia BRO.
event bro_init(){

  print fmt("Hora de inicio: %s", current_time());
```

```
}

## Evento que se genera cuando BRO va a tenerminar, menos si se realiza mediante una lla
event bro_done(){

  ## Mostramos lo que tenemos en la tabla de emparejados
  for(s in emparejados){
    ## print fmt("Tamaño de la fila de la tabla: %d", |empa[s]|);
    ## print fmt("Tenemos: %s en %s a %s en %s", emparejados[s]$id$orig_h, emparejados[s
    ## print fmt(" de %s en %s a %s en %s", s$id$orig_h, s$id$orig_p, s$id$resp_h, s$id$
    informacion_coincidencia(emparejados[s], s);
  }

  ## for(i in emparejados){
    ## print fmt("Tenemos lo siguiente:");
    ## print emparejados[i];
  ## }

  print fmt("Total de flujos: %d", tam);
  print fmt("Hora de finalizacion: %s", current_time());
}
```

En el primer evento solamente mostramos la hora a la que se inicia el programa mediante la función $current_time$. En el evento del final mostramos los flujos que han sido emparejados

# Capítulo 5

# Conclusiones y trabajo futuro

Después de lo visto anteriormente se puede llegar a la conclusión de que mediante un NMS, al cual mediante la agregación de funcionalidades, se le proporciona la capacidad de clasificar el tráfico de una red mediante emparejamiento de flujos. En nuestro caso los protocolos analizados son solamente TCP, UDP e ICMP, es decir, los protocolos de transporte.

Trabajar con Bro al principio fue algo tedioso, pues había que aprender su lenguaje de programación propio y no había mucha documentación disponible, a parte de la que hay en su página web, pero una vez realizadas las primeras tomas de contacto y resueltos los problemas derivados de no conocer bien el lenguaje, como acceder a los datos de las tablas mediante $ en lugar de un punto, como suelo estar acostumbrado, todo fue bastante sencillo.

Fue muy sencillo poder mostrar los datos de los flujos, pues que se disponga de un tipo de dato que almacene todos los datos correspondientes al flujo hace que trabajar con ello sea muy sencillo, incluso mostrar el de todos los flujos que pasan por el programa, y que el rendimiento no se vea afectado. Puede considerarse un inconveniente que no disponga de interfaz, pues en el terminal se puede llegar a confundir las cosas dependiendo de la velocidad con la que saque la información, que sea un lenguaje basado en eventos también puede llegar a ser un problema a la hora de mostrar la información, pues puede ser que estemos mostrando la información de dos flujos que estamos comparando de tipo UDP por ejemplo, y que salte un evento de TCP y se muestre mientras todavía se está mostrando la información del evento de UDP.

Obviamente este programa al solo poder emparejar flujos mediante el pro-

tocolo de transporte se puede quedar corto a la hora de querer obtener un análisis más exhaustivo, de modo que inspeccionemos los paquetes de una forma más profunda, de modo que a este programa se le puede agregar más funcionalidades haciendo que sea más completo, y clasifiquen además los flujos por el tipo de protocolo en la capa de aplicación pertenecen, con esto nos referimos a HTTP, SMTP, FTP, DNS y demás. Esto podría llegar a realizarse analizando más detenidamente los puertos.

A nivel personal me ha gustado poder aprender más sobre redes y cómo analizarlas, pues no es un tema en el cual no tengo unos conocimientos demasiados amplios, aparte de los básicos.

## 5.1. Conclusiones

## 5.2. Trabajo futuro