



TRABAJO FIN DE GRADO  
GRADO EN INGENIERÍA DE TECNOLOGÍAS DE  
TELECOMUNICACIÓN

# Integración de un clasificador de tráfico basado en el tamaño del mensaje en TIE

---

**Autor**

Juan Carlos Martínez de la Torre

**Director**

Jesús Esteban Díaz Verdejo



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

---

Granada, Septiembre de 2015







# Integración de un clasificador de tráfico basado en el tamaño del mensaje en TIE

---

**Autor**

Juan Carlos Martínez de la Torre

**Director**

Jesús Esteban Díaz Verdejo



DEPARTAMENTO DE TEORÍA DE LA SEÑAL, TELEMÁTICA Y  
COMUNICACIONES



# **Integración de un clasificador de tráfico basado en el tamaño del mensaje en TIE**

Juan Carlos Martínez de la Torre

**Palabras clave:** Clasificación de tráfico, análisis del mensaje, capa de transporte, TIE

## **Resumen**

El incremento de la cantidad de tráfico, así como de las distintas tipologías de aplicaciones que se pueden encontrar en la red, ha convertido a la identificación de tráfico en un tarea primordial si se pretende gestionar una red o garantizar la seguridad de los usuarios que la utilizan. La probada ineficacia del método de identificación basado en los puertos de la capa de transporte y los problemas de privacidad asociados a la inspección de la carga útil de los paquetes, ha propiciado que muchos investigadores busquen alternativas, algunas de ellas basadas en características distintivas y de fácil acceso como son aquellos métodos denominados 'ciegos' o no invasivos.

En este trabajo, se propone la implementación de un método, desarrollado previamente por el grupo de investigación del tutor, en base al análisis del tamaño de los mensajes intercambiados en una comunicación, característica que puede obtenerse inspeccionándose únicamente la capa de transporte. Dicho modelo de clasificación resulta robusto contra algunos de los problemas actuales, tales como la ofuscación o el cifrado de la carga útil. Además, permite una detección temprana de las aplicaciones, ya que limita la inspección a los primeros mensajes intercambiados.

La implementación a realizar será integrada dentro de TIE, una plataforma creada recientemente con el objetivo de habilitar la comparación de las diferentes metodologías de identificación de tráfico existentes. El desarrollo de este trabajo se centrará en elaborar un algoritmo de entrenamiento, que permita generar modelos distintivos de cada aplicación, y que a la vez sea compatible con la estructura predefinida de TIE.





# Integración de un clasificador de tráfico basado en el tamaño del mensaje en TIE

Juan Carlos Martínez de la Torre

**Keywords:** Traffic classification, message analysis, transport layer, TIE ....

## Abstract

The rise of the amount of traffic, as well as the increasing number of different applications that we can find on the Internet, has converted traffic identification in a relevant task in order to many network management and to guarantee the security of Internet users. The tested ineffectiveness of the classic method of identification based on the ports of the transport layer and the issues of privacy related to the inspection of the payload of the packets, has motivated several researchers to investigate alternatives, some of them based on distinctive and easily accessible parameters of each application, these methods are known as blind.

In this work, we propose the implementation of a method, which has been recently developed by an investigation group of the tutor, build upon the analysis of the message sizes interchanged in a communication, parameter which can be acquired inspecting just the transport layer. This classification model is specially handy with some of the current problems like obfuscation or encryption of payload. Furthermore, it allows early detection of applications, due to that it just inspects the first messages.

The implementation of the exposed method will be integrated in TIE, a tool which has been recently created with the aim of comparing different methodologies of traffic identification. The development of this work will be focused on elaborating a training algorithm, which enables generating models of different applications, and, at the same time, it has to be concilliable with the predefined structure of TIE.



---

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

**Informan:**

Que el presente trabajo, titulado *Integración de un clasificador de tráfico basado en el tamaño del mensaje en TIE*, ha sido realizado bajo su supervisión por **Juan Carlos Martínez de la Torre**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a    de Septiembre de 2015.

**El director:**

**D. Jesús Esteban Díaz Verdejo**



# Agradecimientos

A mis padres...



# Índice general

	Página
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	2
1.2. Problemática asociada . . . . .	3
1.3. Objetivos . . . . .	4
1.4. Especificaciones . . . . .	5
1.5. Estructura del trabajo . . . . .	6
1.6. Planificación . . . . .	6
1.6.1. Recursos . . . . .	7
1.6.2. Fases de desarrollo . . . . .	7
1.7. Estimación de costes . . . . .	10
1.7.1. Recursos materiales . . . . .	10
1.7.2. Recursos humanos . . . . .	10
<b>2. Antecedentes</b>	<b>13</b>
2.1. DPI (Deep packet inspection) . . . . .	14
2.2. Métodos no invasivos . . . . .	15
2.3. TIE . . . . .	16
<b>3. Identificación del tráfico en base al tamaño de los mensajes</b>	<b>23</b>
3.1. Introducción . . . . .	23
3.2. Extracción de características . . . . .	25
3.2.1. Delimitación del mensaje . . . . .	26
3.2.2. Normalización del tamaño de los mensajes . . . . .	27
3.3. Sistema de clasificación . . . . .	28
3.3.1. Modelado mediante gaussianas múltiples . . . . .	30
3.4. Entrenamiento del modelo . . . . .	32
3.4.1. Algoritmo <i>K-medias</i> . . . . .	33
3.4.2. Cálculo de <i>centroides</i> . . . . .	35
3.4.3. Bisección de clústeres . . . . .	37
3.4.4. Pasos del modelo . . . . .	39

<b>4. Implementación práctica</b>	<b>41</b>
4.1. Arquitectura del clasificador . . . . .	41
4.2. Análisis estructural de TIE . . . . .	42
4.2.1. Inicialización y selección del modo de operación . . . .	43
4.2.2. Elección del medio de entrada . . . . .	45
4.2.3. Bucle principal . . . . .	46
4.2.4. Procesado del paquete . . . . .	47
4.2.5. Clasificación y entrenamiento . . . . .	49
4.2.6. Finalización . . . . .	49
4.3. Parametrización . . . . .	50
4.3.1. Argumentos de entrada para parametrización . . . . .	53
4.3.2. Nuevos parámetros almacenados por sesión . . . . .	53
4.4. Fase de activación . . . . .	56
4.4.1. Clasificador habilitado . . . . .	56
4.4.2. Lectura de modelos . . . . .	57
4.5. Modo de clasificación . . . . .	60
4.5.1. Sesión Clasificable . . . . .	60
4.5.2. Clasificación de sesión . . . . .	61
4.6. Modo de entrenamiento . . . . .	64
4.6.1. Inicio del proceso . . . . .	65
4.6.2. Almacenamiento de parámetros . . . . .	67
4.6.3. Ejecución del algoritmo de clustering . . . . .	68
4.6.4. Acciones finales del entrenamiento . . . . .	90
<b>5. Pruebas y simulaciones</b>	<b>93</b>
5.1. Extracción de parámetros para un flujo SMTP . . . . .	93
5.2. Simulación de clasificación . . . . .	96
5.3. Simulaciones variando variables que regulan el sistema . . . .	101
5.3.1. Número de gaussianas permitido y número de mensajes iniciales necesarios por sesión . . . . .	101
5.3.2. Tamaño mínimo de carga útil . . . . .	106
5.3.3. Valor utilizado para escalar el tamaño de cada mensaje	108
<b>6. Conclusiones y futuro trabajo</b>	<b>109</b>
6.1. Trabajo futuro . . . . .	110
<b>Apéndices</b>	<b>112</b>
<b>A. Extracción de los modelos externos</b>	<b>115</b>
A.1. Análisis del formato del modelado . . . . .	115
A.2. Procesado de cada uno de los ficheros . . . . .	117
A.3. Creación del fichero <i>foreign_signature_file.txt</i> . . . . .	117
A.4. Simulación de clasificación con este modelado . . . . .	119



<b>B. Archivo de configuración <i>MSBC_conf.c</i></b>	<b>121</b>
B.1. Propósito de cada variable . . . . .	121
<b>C. Procesado del tráfico externo</b>	<b>125</b>
C.1. Identificación de los protocolos presentes . . . . .	125
C.1.1. Análisis de los campos útiles . . . . .	125
C.1.2. Obtención de las aplicaciones presentes . . . . .	127
C.2. Transformar al formato reconocido por TIE . . . . .	129



# Índice de figuras

	Página
1.1. Evolución de la cantidad de tráfico generado en Internet . . .	3
1.2. Diagrama de Gantt: De Noviembre a Marzo . . . . .	9
1.3. Diagrama de Gantt: De Abril a Agosto . . . . .	9
2.1. Evolución del interés por el método de clasificación . . . . .	14
2.2. Bloques lógicos de TIE [13] . . . . .	18
3.1. Funcion de escalado . . . . .	27
3.2. Distancia escalada . . . . .	28
3.3. Módulos del clasificador . . . . .	29
4.1. Principales bloques de TIE . . . . .	43
4.2. Bloques lógicos de TIE: Inicializacin y selección del modo de operación . . . . .	44
4.3. Bloques lógicos de TIE: Medios de entrada . . . . .	45
4.4. Bloques lógicos de TIE: Bucle principal . . . . .	47
4.5. Bloques lógicos de TIE: Procesado del paquete . . . . .	48
4.6. Bloques lógicos de TIE: Entrenamiento y clasificación . . . .	50
4.7. Bloques lógicos de TIE: Finalización . . . . .	51
4.8. Esquema del procedimiento de extracción del mensaje . . . .	55
4.9. Estructura de almacenamiento del modelo para cada aplicación	58
4.10. Extracto del fichero usado para asociar la identidad de cada aplicación con el conjunto de identificadores de TIE . . . . .	59
4.11. Procedimiento de clasificación . . . . .	62
4.12. Estructura de almacenamiento de modelos para cada aplicación	66
4.13. Fragmento del fichero <i>TIE_app.txt</i> . . . . .	67
4.14. Archivo <i>own_MSBC_map_file.txt</i> . . . . .	68
4.15. Extracto de un fichero de recogida de muestras para el entre- namiento . . . . .	68
4.16. Estructura usada para almacenar las muestras de una aplicación	69
4.17. Arquitectura previa al algoritmo de agrupamiento . . . . .	71
4.18. Estructura de selección de cada espacio muestral para una sola aplicación . . . . .	72

4.19. <i>struct clusters_trained</i> . Estructura temporal usada durante el algoritmo de <i>clustering</i> , empleada en almacenar las diferentes distribuciones establecidas durante dicho proceso . . . . .	73
4.20. Arquitectura del procedimiento de asignación de cada muestras al <i>clúster</i> con el <i>centroide</i> más cercano . . . . .	77
4.21. Estructura del procedimiento empleado para el cálculo del <i>centroide</i> de un <i>clúster</i> dado . . . . .	79
4.22. Estructura del proceso de estimar la varianza para los distintos <i>clusters</i> . . . . .	81
4.23. Estructura del procedimiento seguido para el cálculo del error cuadrático medio total . . . . .	83
4.24. Inicio del algoritmo de <i>clustering</i> . . . . .	85
4.25. Bucle de bisección de <i>clústeres</i> . . . . .	87
4.26. Algoritmo K-medias básico . . . . .	90
5.1. Intercambio de tramas para un flujo SMTP . . . . .	94
5.2. Fichero <i>ground_file.tie</i> generado por la pre-clasificación de un flujo SMTP . . . . .	95
5.3. Almacenamiento de muestras para un flujo SMTP . . . . .	96
5.4. Captura parámetros por defecto en una ejecución del clasificador . . . . .	99
5.5. Estadísticas del tráfico procesado . . . . .	99
5.6. Matriz de aciertos para una simulación con los valores por defecto de sus variables . . . . .	100
5.7. Porcentaje de identificaciones correctas en función del número de mensajes analizados por conexión y del máximo número de gaussianas por modelo . . . . .	102
5.8. Resultados de una simulación para un solo mensaje analizado por comunicación . . . . .	104
5.9. Porcentaje de sesiones correctamente identificadas en función del mínimo tamaño de carga útil del paquete para que no acabe el mensaje actual . . . . .	107
5.10. Porcentaje de sesiones correctamente identificadas en función del mínimo tamaño de carga útil del paquete para que no se de por evaluado el mensaje . . . . .	107
A.1. Extracto de un fichero de almacenamiento de modelos generados por la aplicación externa a TIE . . . . .	116
A.2. Salida generada al aplicar el código de comandos <i>beta.awk</i> sobre <i>17_1_C-cluster.csv</i> . . . . .	117
A.3. Estructura de bucles enlazados en el fichero <i>scriptFormatos.sh</i> . . . . .	118
A.4. Archivo <i>foreign_MSBC_map_file.txt</i> , habilita la asociación entre la identificación de cada modelo con sus números identificativos de TIE <i>app_id</i> y <i>sub_id</i> . . . . .	119

A.5. Resultados de una simulación usando el modelado generado por la aplicación externa a TIE . . . . .	119
B.1. Estructura de parámetros del sistema . . . . .	121
C.1. Fichero con el etiquetado de cada flujo del tráfico usado para el entrenamiento . . . . .	126
C.2. Análisis de un intercambio de paquetes para un flujo del tráfico usado durante el entrenamiento . . . . .	127
C.3. Extracto del fichero All_protocols.txt . . . . .	128
C.4. Contenido del fichero protocols_found.txt . . . . .	128
C.5. Contenido del fichero protocols_mapped.txt . . . . .	129
C.6. Estructura de lista enlazadas usada para almacenar la asociación entre las identidades de TIE y las identidades usadas por el archivo de preclasificación externo . . . . .	129
C.7. Fichero ya transformado al formato de fichero de preclasificación reconocible por TIE . . . . .	130



# Índice de Tablas

	Página
1.1. Estimación del tiempo empleado en el desarrollo del trabajo .	10
1.2. Coste asociado a los recursos materiales . . . . .	11
1.3. Estimación del coste de los recursos humanos empleados . .	11
2.1. Comandos de TIE para el filtrado de paquetes . . . . .	18
2.2. Comandos de TIE para indicar los parámetros a extraer . . .	19
2.3. Argumentos de TIE dedicados a organizar la salida de resultados	20
2.4. Comandos de TIE relacionados con el entrenamiendo de un <i>plugin</i> . . . . .	20
2.5. Argumentos generales de TIE . . . . .	21
4.1. Ejemplos de primitivas de filtrado BPF . . . . .	46
4.2. Argumentos añadidos a TIE . . . . .	53
4.3. Parámetros añadidos a cada sesión . . . . .	54
5.1. Comandos usados para pre-clasfición con TIE al aplicar el clasificador l7_1.3 . . . . .	94
5.2. Argumetnos usados para el entrenamiento del clasificador pro- puesto en un flujo SMTP . . . . .	95
5.3. Sesiones presentes en el archivo de tráfico externo . . . . .	97
5.4. Principales variables del entorno de simulación . . . . .	97
5.5. Argumentos usados para el entrenamiento del <i>plugin</i> MSBC_1.1	98
5.6. Estadísticas de atribuciones para 1 y 5 mensajes . . . . .	105
5.7. Estadísticas de identificación y precisión para cada aplicación para 1 y 4 mensajes . . . . .	106
A.1. Atribución del número de identidad para los clústeres externos	116
B.1. Valores por defecto para las variables globales del clasificador MSBC . . . . .	123





# Capítulo 1

## Introducción

La identificación del tráfico circulante en una red, entendida como la atribución de los flujos o paquetes a la aplicación que los genera [1], constituye un problema de enorme interés en las actuales redes. Sin embargo, durante los últimos años, clasificar el tipo de tráfico que discurre por una red se ha convertido en una de las problemáticas más desafiantes en el ámbito de las redes de telecomunicaciones.

La identificación de tráfico es esencial si se pretende entender el funcionamiento y uso de una red, permitiendo adaptar las redes al tráfico que soportan, ofreciendo un mejor servicio a los millones de clientes que usan Internet cada día. Además, el aumento de la capacidad y de la disponibilidad mediante las conexiones de banda ancha han dado lugar a un complejo escaparate donde cualquier usuario genera diferentes tipologías de tráfico, distando mucho del usuario básico en el que se pensó cuando se diseñó Internet.

Tradicionalmente, en las redes TCP/IP como Internet, se ha utilizado el número de puerto de la capa de transporte como indicativo de la aplicación que había generado un flujo de datos. Sin embargo, es bien conocido que actualmente esta forma de proceder ha quedado obsoleta [2], por lo que investigación en torno a nuevas metodologías de identificación de tráfico se ha convertido en una importante línea de trabajo.

Aunque se pueden usar otras aproximaciones, una alternativa a este método son aquellos que buscan características distintivas de cada aplicación en la capa de transporte o en el comportamiento del flujo generado en la red, métodos que pueden hacer frente a algunos de los retos que presenta la identificación de tráfico hoy día, tales como el tunelado o la ofuscación.

En este trabajo se pretende implementar un identificador de tráfico, desarrollado recientemente [3], basado en la caracterización de los flujos por el tamaño de los mensajes intercambiados, para su incorporación en una herramienta de identificación de tráfico de dominio público preexistente. En este contexto, se entenderá por flujo la comunicación entre dos entidades a

nivel de transporte caracterizada por la tupla formada por las direcciones IP origen y destino, los puerto de origen y destino y el protocolo de transporte utilizado (UDP o TCP). En este sentido, es necesario indicar que se ha extendido el concepto de flujo utilizado habitualmente para incorporar las conexiones UDP.

## 1.1. Motivación

La identificación de tráfico pertenece al amplio número de herramientas que son de utilidad en las operaciones de gestión, administración y mantenimiento (OAM) de redes. Las dos aplicaciones más inmediatas de la identificación de tráfico están relacionadas con la ingeniería de tráfico y la seguridad. En particular, se puede utilizar en la asignación de recursos de una red TCP/IP, así como para mejorar la fiabilidad de los sistemas de detección de intrusos (NIDS), ya que posibilita detectar tráfico anómalo si no se respetan los patrones establecidos por la caracterización de tráfico previa. Algunos de los ámbitos en los que es útil son los siguientes:

- Cuando se desarrolló Internet se orientó para dar un servicio "best-effort", es decir, se procura el mejor servicio posible a todos y cada uno de los flujos de datos sin atender a si es una llamada vía red, con sus correspondientes requisitos de tiempo de entrega, o si se trata de un simple correo electrónico. Resulta evidente, que este no es el método más adecuado de organizar la entrega de tramas, por ello es que en la última década se han generado diferentes mecanismos para superar este problema basados en la diferenciación de servicios [4].

Es aquí donde entra el concepto de identificación de tráfico, pues es fundamental una correcta detección de aplicaciones para poder asignar prioridades diferentes a los flujos. Por otro lado, cualquier diseñador de una futura red debe hacer un estudio exhaustivo del tráfico futuro que la red va a soportar o del que soporta actualmente, si se trata de una remodelación.

- Si un método de caracterización de tráfico fuera realmente fiable y exacto, se podría usar para evaluar variaciones del tipo tráfico asociado a una aplicación, previamente caracterizada, permitiendo detectar anomalías, lo que puede ser considerado como un tráfico no legítimo y por lo tanto malintencionado

Un problema añadido a la no identificación correcta del tráfico es que todo uso legítimo que conlleve una variación significativa del tráfico generado también será interpretado como tráfico anómalo y por lo tanto rechazado [5].

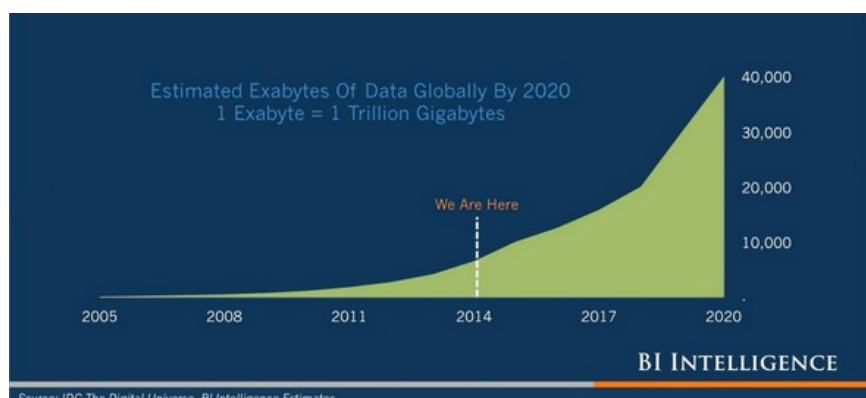


Figura 1.1: Evolución de la cantidad de tráfico generado en Internet

Consecuentemente, una correcta asociación entre los flujos y las aplicaciones que los generan, nos puede llevar a un aumento en la seguridad de las redes, así como a una calidad de servicio adaptada a las necesidades de la comunicación.

## 1.2. Problemática asociada

Una vez expuestos los beneficios de un método de clasificación eficaz, comentaremos los principales inconvenientes que han propiciado que no se tenga ningún procedimiento sea totalmente aceptado. En este sentido, hemos indicar que, como se expone en el Capítulo 2, la aproximación más exitosa en la actualidad se basa en la inspección de paquetes, esto es, en el análisis de las cargas útiles de los paquetes. Sin embargo, esta solución dista de ser la más adecuada por estas razones:

- El aumento vertiginoso de datos generados en los últimos años, como se ve en la Figura 1.1, así como el número de dispositivos (por ejemplo tablets, telefonos móviles, relojes inteligentes etc...) que acceden a Internet, hace cada vez más amplio el número de aplicaciones que necesitan ser identificadas.
- Uno de los principales dilemas derivadas de la inspección de paquetes es garantizar la privacidad de los datos emitidos por los usuarios, lo cual limita significativamente el proceso.

Problema similar al anterior es el que sucede cuando se usan protocolos de encriptación, pues cuando los paquetes están protegidos (por ejemplo IPsec) es imposible realizar una inspección completa.

- Los procedimientos de clasificación de tráfico deben ser diseñados para ser utilizados sobre enlaces de alta capacidad, siendo necesario que la

metodología este lo más optimizada posible para poder soportar toda esa carga de procesamiento de tráfico en tiempo real. Este problema aumentará en el futuro, al ser mayor la cantidad de tráfico a soportar, tal y como vimos antes.

- Los protocolos P2P, cuyo uso ha aumentado durante los últimos años, se han ido convirtiendo en los dominadores de los intercambios de información en numerosos escenarios. Las aplicaciones que usan estos protocolos (por ejemplo *bittorrent*, *e-mule*) suelen usar los puertos de forma aleatoria, además de utilizar ofuscación forma habitual, esto es, generan estructuras de comunicación diferentes a las asociadas comúnmente a estos protocolos.

Los algoritmos de identificación de tráfico deben ser capaces de no ralentizar el tráfico, aún estando en enlaces de alta capacidad. Por ello, es necesaria la optimización de dichos algoritmos, ya que deberán de soportar grandes cantidades de datos con una capacidad limitada de cálculo, posibilitando la utilización efectiva de la identificación de tráfico en escenarios reales.

El método que se va a implementar está basado en el expuesto en el informe [3], donde se propone un método de identificación de tráfico basado en la caracterización de las aplicaciones por el tamaño de sus mensajes, un procedimiento que es bastante robusto ante los problemas antes expuestos.

### 1.3. Objetivos

El principal objeto del presente trabajo es implementar el algoritmo de identificación de tráfico expuesto en el documento [3] sobre TIE, una plataforma destinada a la investigación sobre sistemas de clasificación de tráfico, permitiendo así la posible comparación de la técnica aquí implementada con otras diferentes. Esta implementación se puede dividir en tres partes:

- Extracción de los parámetros necesarios para la clasificación. El método propuesto se basa en los mensajes intercambiados en una conexión, por lo que hace falta implementar los mecanismos necesarios para la extracción de dichos mensajes de cualquier flujo.
- Clasificación de los flujos. El clasificador estará basado en el uso de probabilidades, definidas por gaussianas, sobre una secuencia de mensajes dada. Cada secuencia de mensajes será en sí misma una secuencia de estados similar a un proceso de Markov.
- Elaboración del algoritmo de entrenamiento. El proceso de clasificación hace uso de un modelado de las distintas aplicaciones, basado en distribuciones normales. Este modelado deberá de ser generado por el

propio clasificador en una fase previa de entrenamiento, haciendo uso de diferentes flujos pre-clasificados.

Es en el entrenamiento el ámbito que más desarrollará durante este trabajo, pues en el documento que expone el método de clasificación [3] se especifica que se usará el algoritmo de K-medias, pero sin hacer demasiadas indicaciones. Se definirán distintos parámetros que regulen la ejecución de este algoritmo con objeto de evaluar la configuración que de mejores resultados.

Por último, como se pretende que el clasificador sea compatible con TIE, se deberá de respetar el modelo que ofrece TIE para la implementación de las diferentes técnicas de clasificación, ya que esta plataforma define de forma predeterminada los bloques lógicos en los que se dividirá el sistema que se quiere implementar.

Finalmente, se realizarán una serie de simulaciones que corroboren la efectividad de la técnica implementada, así como la correcta implementación e integración en TIE de los diferentes bloques lógicos necesarios.

## 1.4. Especificaciones

La identificación de tráfico es el proceso por el que se reconoce la aplicación que ha generado un flujo dado. Por lo tanto, es en el concepto de flujo donde radica el principio del método que se pretende implementar. Normalmente se conoce a un flujo por la tupla formada por:

- Dirección IP del emisor
- Dirección IP del receptor
- Puerto del emisor
- Puerto del receptor
- Protocolo utilizado en la capa de transporte UDP/TCP

Siendo las referencias al emisor y al receptor intercambiables en una conexión, en función de la dirección.

Estos flujos pueden ser tomados desde archivos *pcap*, trabajando el sistema en modo *offline*, o directamente desde una interfaz de red, en modo tiempo real. El conjunto de pruebas para el desarrollo del algoritmo aquí implementado se ha realizado únicamente en modo *offline*.

Para la clasificación de flujos, es necesario que previamente se tengan caracterizadas a cada una de las aplicaciones que son identificables. Esta caracterización será generada durante la fase de entrenamiento, en la cual se tomarán muestras, de secuencias de mensajes, a partir de las que se crearán

dichos modelos. Finalmente, los modelos de las aplicaciones estarán contenidos en un solo fichero, que servirá como referencia durante el proceso de clasificación.

Este clasificador deberá estar incorporado a la plataforma TIE (*Traffic Identification Engine*) desarrollada por la Universidad de Nápoles. Esta aplicación define el conjunto de bloques lógicos en los que se debe dividir el clasificador, como son el de inicialización o el de recopilación de datos para el posterior entrenamiento.

El programa será generado en lenguaje C, siendo compatible con las diferentes plataformas con las que TIE es compatible como Linux o MacOS.

## 1.5. Estructura del trabajo

La presente memoria de trabajo esta dividida en 6 capítulos, que a su vez están formados por diferentes apartados. Seguidamente, se expone el título de los diferentes capítulos, acompañado de una breve descripción:

1. **Introducción.** En este capítulo se exponen la motivación y la organización de este trabajo.
2. **Antecedentes.** Contiene la evolución de las diferentes técnicas de identificación de tráfico
3. **Identificación del tráfico en base al tamaño de los mensajes.** Se expone de forma teórica el método que se pretende implementar
4. **Implementación práctica.** Se describe la implementación generada, así como la justificación de las diferentes decisiones tomadas durante el desarrollo del trabajo.
5. **Pruebas y simulaciones.** Se comentan las simulaciones realizadas para la verificación del algoritmo.
6. **Conclusiones y futuro trabajo** Se detallan las conclusiones del trabajo y las posibles mejoras en el sistema implementado.

## 1.6. Planificación

En esta sección se pretenden exponer los costes y la planificación asociada a la realización del trabajo.

En primer lugar, se enumerarán los diferentes recursos empleados en el desarrollo del trabajo, tanto humanos como materiales. Posteriormente, se comentará el tiempo empleado, así como la planificación seguida elaborándose para ello un Diagrama de Gantt o cronograma.

### 1.6.1. Recursos

#### 1.6.1.1. Recursos humanos

- D. Jesús Esteban Díaz Verdejo, profesor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, en calidad de tutor del trabajo de fin de grado.
- D. Juan Carlos Martínez de la Torre, alumno de la Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones de la Universidad de Granada, como autor del trabajo.

#### 1.6.1.2. Hardware

- Ordenador personal.
- Impresora.

#### 1.6.1.3. Software

- Sublime Text. Editor de texto y de código fuente sobre el que se desarrollarán la programación en C.
- Valgrind. Conjunto de herramientas de ayuda para la depuración de problemas de memoria de los programas.
- TIE. Plataforma para la cual se pretende diseñar el clasificador.
- Gnuplot. Herramienta de dibujo de gráficas.
- Dia. Aplicación para la creación de diagramas.
- Wireshark. Aplicación útil para analizar tráfico de red.
- Texmaker. Editor de documentos LaTeX.
- Ubuntu versión 14.04. Sistema operativo usado para el desarrollo del trabajo.

### 1.6.2. Fases de desarrollo

El trabajo realizado se divide en un total de 9 tareas, que van desde la recopilación de datos sobre la temática que se va a tratar hasta el procesamiento del tráfico que se usará para el entrenamiento del sistema.

#### 1.6.2.1. Tarea 1: Revisión de antecedentes

Se analiza el panorama actual en torno a la identificación de tráfico, profundizando en los objetivos para los que ha sido creada la plataforma TIE.

**1.6.2.2. Tarea 2: Exploración de la estructura de TIE**

Se investiga de forma exhaustiva los distintos bloques que forman TIE, con el objetivo de recopilar la mayor información posible, ya que esto será de ayuda para la implementación del clasificador.

**1.6.2.3. Tarea 3: Diseño e implementación del método de extracción de mensajes**

Se define la estructura del mecanismo que se encargará de analizar los flujos de datos, generando la secuencia de mensajes asociada a ese flujo.

**1.6.2.4. Tarea 4: Diseño e implementación de lector del modelado**

Se estudia como será el archivo que almacenará los diferentes modelos de las aplicaciones, definiéndose una estructura fija para este archivo, así como para el lector de este archivo.

Se extraen los primeros modelos de un conjunto de archivos con modelos definidos por otra aplicación.

**1.6.2.5. Tarea 5: Implementación del clasificador**

Se genera el código que llevará a cabo la clasificación.

**1.6.2.6. Tarea 6: Diseño e implementación del entrenamiento**

Se profundiza acerca del tipo de entrenamiento que se va a programar. En primer lugar, se diseña e implementa el algoritmo de agrupamiento (*clustering*) y se comprueba que funciona sin estar integrado en el clasificador. Posteriormente, se integra en el clasificador y se verifica que los modelos generados son coherentes.

**1.6.2.7. Tarea 7: Diseño del procesado del tráfico pre-clasificado externo**

Actualmente, la plataforma TIE no cuenta con demasiados clasificadores implementados, por lo que se necesita tráfico preclasificado externo para llevar a cabo las simulaciones de funcionamiento del sistema.

**1.6.2.8. Tarea 8: Simulaciones y pruebas**

Se realizan diferentes pruebas con el objetivo de comprobar el correcto funcionamiento del sistema implementado.

**1.6.2.9. Tarea 9: Redacción del trabajo**

Se redacta el presente documento.



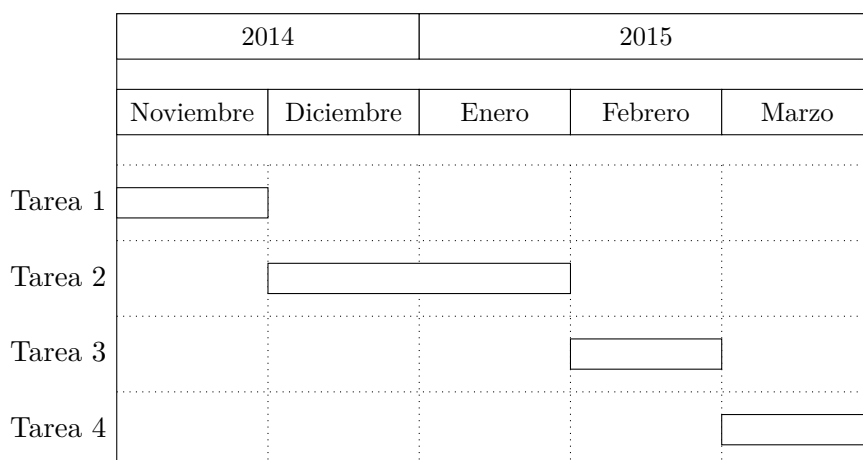


Figura 1.2: Diagrama de Gantt: De Noviembre a Marzo

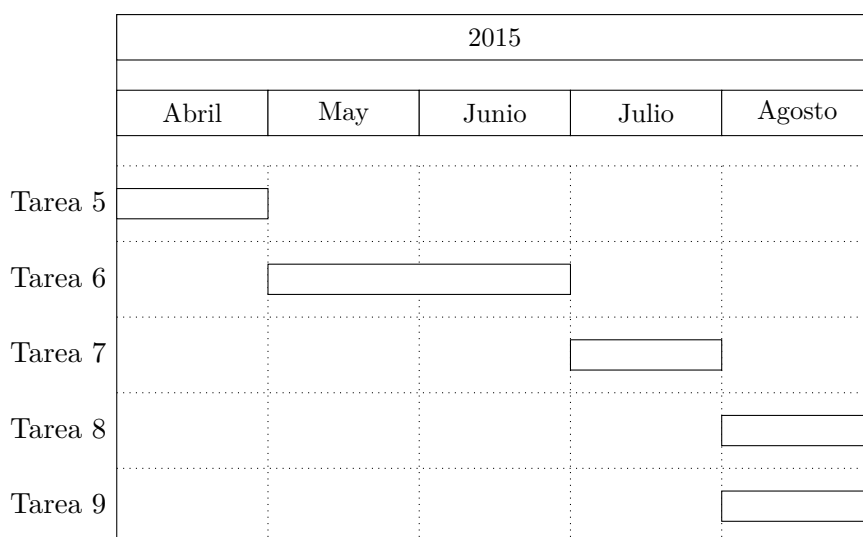


Figura 1.3: Diagrama de Gantt: De Abril a Agosto

#### 1.6.2.10. Planificación temporal

La planificación temporal seguida queda reflejada en los dos diagramas de Gantt de las Figuras 1.2 y 1.3.

Una estimación del tiempo empleado en la ejecución de las diferentes tareas viene expuesto en la tabla 1.1.

Para cada tarea se han tenido en cuenta los días de los fines de semana para los meses en los que se desarrollo dicha tarea si ha sido en periodo lectivo, ya que durante la semana no se trabajó en el proyecto. Para las tareas ejecutadas en verano se han tomado el total de todos los días, excepto en los que no se ha desarrol el proyecto.

El horario ha sido de 8 horas por día, excepto durante el mes de agosto, que fueron un total de diez horas repartidas entre las dos tareas ejecutadas.

Tiempo empleado en el trabajo			
Tarea	Días empleados	horas/día	Total de horas
Tarea 1	10	8	80
Tarea 2	30	8	240
Tarea 3	8	8	64
Tarea 4	8	8	64
Tarea 5	8	8	64
Tarea 6	16	8	128
Tarea 7	16	8	128
Tarea 8	25	5	125
Tarea 9	25	5	125
Horas empleadas en total			1018

Tabla 1.1: Estimación del tiempo empleado en el desarrollo del trabajo

## 1.7. Estimación de costes

### 1.7.1. Recursos materiales

Durante el trabajo se ha hecho uso de dos ordenadores portátiles, cuyo coste asociado al empleo de cada uno de estos terminales viene expuesto en la Tabla 1.2. Para la elaboración de esta tabla se ha tomado que el tiempo de vida medio de un equipo a rendimiento medio es de dos años, lo que son un total 3840 horas de trabajo:

$$2\text{años} * 12 \frac{\text{meses}}{\text{año}} * 20 \frac{\text{días}}{\text{mes}} * 8 \frac{\text{horas}}{\text{día}} = 3840\text{horas}$$

Tomado como referencia dicho número de horas como el máximo de tiempo que se puede emplear un equipo, se calculará el coste debido al uso de cada equipo en este trabajo.

### 1.7.2. Recursos humanos

Para la estimación de costes del empleo de recursos humanos se ha tomado que la hora invertida por el alumno valdrá 20€, mientras que la empleada por el tutor del proyecto es de 50€.

Con respecto al tiempo empleado, se toma la estimación realizada para el alumno en la sección 1.6.2.10, para el tutor se toma que se han realizado un total de 20 horas de trabajo repartidas entre horas de tutoría y las necesarias para la preparación del material.

<b>Coste asociado a los recursos materiales</b>			
Equipo	Coste del equipo (€)	Horas de uso (h)	Coste asociado (€)
1º Portátil	500	800	104
2º Portátil	900	218	61
Coste asociado total			165 €

Tabla 1.2: Coste asociado a los recursos materiales

<b>Coste asociado a los recursos humanos</b>		
Horas empleadas (h)	Precio de cada hora (€/h)	Coste asociado (€)
1018	20 €	20360
20	50 €	1000
Coste de recursos humanos		21360

Tabla 1.3: Estimación del coste de los recursos humanos empleados

Por lo tanto, realizando la suma del coste proveniente de los recursos humanos y del coste asociado a los recursos materiales, se tiene que el coste total es de 21525, VEINTIUNO MIL QUINIENTOS VEINTICINCO EUROS



## Capítulo 2

# Antecedentes

Históricamente, el método de identificación utilizado para reconocer qué aplicación es transportada por cada flujo consistía, básicamente, en asociar el número de puerto de la capa de transporte con su correspondiente aplicación, de acuerdo a lo registrado en la IANA [6]. Sin embargo, durante la última década han surgido diferentes problemas en torno a este método que se pueden resumir en los siguientes [7]:

- La irrupción de aplicaciones que no tienen un puerto registrado en la IANA. Además, se observa cómo nuevas aplicaciones hacen uso de puertos asignados previamente a otras para así ocultarse y/o conseguir traspasar el filtrado de los cortafuegos.
- Por otra parte, la masificación de Internet trae como consecuencia la falta de direcciones IP y el uso de NAT, entre otras técnicas, para solucionarlo. Como resultado, se producen asignaciones de puertos que enmascaran los asignados por la IANA para los diferentes servicios, multiplexándose direcciones IP y puertos para diferentes equipos y/o servicios.
- Debido al desarrollo del acceso web, el protocolo HTTP ha empezado a ser empleado como método de encapsulación de otro tipo de tráfico, como mensajería instantánea o contenidos multimedia[8].
- Actualmente, gran parte del tráfico que recorre la red es del tipo P2P. Es conocido que este tipo de tráfico escoge el puerto de forma aleatoria, lo cual agrava el problema.

No obstante, el modelo de clasificación basado en el puerto sigue siendo usado como primera aproximación hoy en día, debido a su simplicidad y rapidez ya que solo se tiene que controlar un campo de la cabecera de cada paquete.

Consecuentemente, la identificación de tráfico es uno de los temas de investigación de mayor relevancia en el campo al que la telemática se refiere.

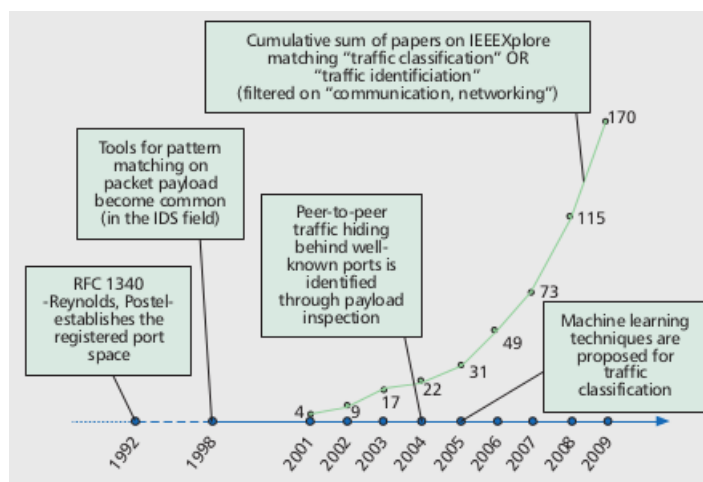


Figura 2.1: Evolución del interés por el método de clasificación

La imagen de la Figura 2.1 refleja el interés de esta temática en el ámbito investigador, que ha aumentado de manera exponencial quedando plasmado en el número de trabajos publicados en los últimos años.

Dado el elevado número de métodos propuestos, únicamente se expondrán los más significativos, haciendo especial énfasis en DPI (Deep packet inspection), el cual presenta mayor precisión, y los métodos no invasivos, ya que el método que se implementará forma parte de ese conjunto.

## 2.1. DPI (Deep packet inspection)

Este método compara el contenido del paquete con firmas de tráfico de aplicaciones previamente caracterizadas, siendo considerado actualmente el mejor y más eficaz. La inspección de la carga útil de un paquete es considerada una técnica realmente fiable, pero tiene una serie de limitaciones impuestas por las condiciones a las que un motor de identificación de tráfico ha de enfrentarse.

Alguno de los problemas presentes en este método son:

- La gran cantidad de tráfico que debe ser analizado en los diferentes escenarios de interés, como son los enlaces de banda ancha, limita el número de firmas que pueden ser utilizadas a fin de evitar sobrecarga en los analizadores.
- Acceder a los datos del usuario es ilegal en muchos países. Además, actualmente muchos protocolos hacen uso de encriptación, imposibilitando una correcta inspección.
- La significativa complejidad que se tiene si se requiere la obtención de

firmas de las distintas aplicaciones, ya que muchas de ellas no tienen una estructura de paquete registrada y porque surgen constantemente nuevas aplicaciones.

Estas dificultades en la implementación de lo que se conoce como DPI ha propiciado la investigación de nuevas metodologías basándose en elementos que sean capaces de caracterizar el tráfico.

Es aquí donde entran las posibles innovaciones, como el uso de este método junto con la identificación basada en puertos[9]. En este procedimiento se limitaba la inspección a los primeros paquetes disminuyéndose así la carga computacional para la clasificación de cada flujo.

Una herramienta de código abierto que implementa este clasificador basado en la inspección del paquete es *L7-filter* que ya está desactualizado, siendo *nDPI* el proyecto que ha tomado el relevo estando todavía en continuo desarrollo.

## 2.2. Métodos no invasivos

Junto a DPI, otros métodos se han desarrollado a lo largo de la última década con el objetivo de evitar la inspección completa del paquete. Basándose en la caracterización de un flujo dado, se pretende la obtención de parámetros representativos de cada aplicación en los flujos que producen, sin necesidad de analizar la carga útil, por lo que se les denomina métodos no invasivos o de clasificación ciega.

La principal ventaja con respecto a DPI es que no se necesita inspeccionar la carga útil, por lo que los parámetros necesarios se obtienen de la capa de transporte. Algunos de los elementos en los que estos métodos se basan son:

- Duración de la comunicación.
- Número de paquetes.
- Dirección de los paquetes upstream/downstream.
- Tamaño medio de los paquetes.
- Intervalo de tiempo entre la llegada de los paquetes.

En general, estos métodos se basan en la parametrización de los flujos para su posterior clasificación, mediante modelos estadísticos o técnicas de aprendizaje automático [10]. Por ello, tanto la selección de los parámetros a utilizar así como la técnica de modelado son los elementos en los que nos debemos centrar si se pretende elaborar un método de identificación de tráfico no invasivo.

En la búsqueda de los elementos que caractericen a las aplicaciones debemos evitar aquellos que puedan verse modificados por componentes dinámicos de la red, como puede ser la posible congestión en los enlaces. Cuanto

menos dependa de factores externos a la aplicación, más robusto será el método. Aún así, algunos estudios han demostrado que esos elementos temporales pueden llegar a ser significativos de cada aplicación [11], mientras que otros estudios defienden lo contrario.

Un conjunto de métodos que hace uso de las métricas obtenidas a través de la caracterización de los flujos son las máquinas de aprendizaje automático. Estos procedimientos cuentan con la ventaja de crear de forma automática los parámetros característicos del flujo de cada aplicación así como de mejorar dichos parámetros a lo largo de su ejecución [10].

A pesar de los numerosos esfuerzos realizados en las diversas investigaciones y del éxito en muchas de ellas, todavía no se dispone de un método que responda a los problemas expuestos anteriormente.

Un problema adicional con respecto a estos sistemas de identificación de tráfico es que algunos necesitan que acabe el flujo para poder identificar la aplicación [12]. Esto impide que la identificación de tráfico pueda realizarse en tiempo real, imposibilitando a la identificación de tráfico como instrumento de NIDS. Ante esto, es necesario buscar un procedimiento que permita reconocer la aplicación utilizando el mínimo número de tramas iniciales.

El método utilizado en este trabajo y que se detallará en el siguiente capítulo se incluye dentro de este amplio grupo, ya que basa la clasificación en el estudio del tamaño de los mensajes iniciales del flujo, lo cual se extrae directamente de la capa de transporte. El modelado de los protocolos se realiza mediante distribuciones de probabilidad de observación de un tamaño de mensaje en una posición concreta del flujo, siendo incorporado a TIE, una plataforma de comparación de diferentes clasificadores.

## 2.3. TIE

Esta herramienta, desarrollada por un grupo de investigadores de Nápoles, es un proyecto para la identificación de aplicaciones a través del análisis del tráfico de la red, nacido con el objetivo de ser un elemento de unión entre diferentes investigadores para aunar conocimiento y comparar el comportamiento de diferentes tipos de clasificadores. [13]

Ofrece la posibilidad de combinar diferentes tipos de clasificadores, permitiendo definir diversas métricas de combinación para la clasificación de flujos. De esta forma, está orientado a que varios clasificadores puedan ejecutarse de forma simultánea. A cada módulo clasificador se le denomina “*plugin*” y, como se ha mencionado.

TIE define cada una de las posibles aplicaciones a clasificar mediante la asignación una identidad, asociándolas posteriormente a un posible grupo. Este tipo de relación permite la comparación entre diferentes motores de clasificación con diferentes granularidades, esto es, por ejemplo, agrupando a distintas aplicaciones como Bittorrent, Gnutella o Edonkey dentro del



grupo de P2P.

Por otra parte, también se definen subaplicaciones, es decir, además del identificador primario de aplicación se le asigna un sub-identificador dentro de la propia aplicación, permitiendo distinguir entre diferentes protocolos o funcionalidades de una misma aplicación, como para FTP entre un flujo FTP de control o un flujo FTP de envío de datos o *stream*.

Los flujos a clasificar pueden ser proporcionados mediante archivos en formato *pcap* o también en tiempo real, capturando los paquetes de un dispositivo de red.

Por otro lado, TIE dispone de diferentes modos de ejecución, los cuales dotan a la plataforma de gran versatilidad. Estos modos son:

- **Modo *offline*:** Cada uno de los flujos es clasificado al final de la ejecución de TIE. Este modo es de especial interés para la puesta a punto de un clasificador en fase de desarrollo y/o de entrenamiento.
- **Modo en tiempo real:** Cada uno de los flujos es clasificado en cuanto se disponga de la información necesaria para llevar a cabo esta clasificación. Este modo es el ideal, ya que puede implementarse en entornos reales para aplicar políticas de calidad de servicio, *QoS*.
- **Modo cíclico:** El tráfico es clasificado cada cierto intervalo de tiempo. Este modo es útil para la extracción de informes periódicos sobre el tráfico que fluye por el enlace bajo análisis.

Dispone además tanto de modo de clasificación como de modo de entrenamiento. Para llevar a cabo el entrenamiento eficaz de un clasificador es necesario disponer de un conjunto de flujos previamente clasificados. Desde el punto de vista del desarrollo y evaluación de clasificadores, esta capacidad supone una de las principales ventajas de TIE, ya que facilita esta tarea al permitir usar como fichero de referencia la clasificación de dichos flujos, generada dicha clasificación por otro *plugin*.

Una de las principales características de TIE es la alta modularidad con la que se ha sido diseñada. Cada *plugin* es independiente de los demás, permitiendo la ejecución individual de cada uno de ellos o, por el contrario, la ejecución conjunta de varios para obtener un resultado de clasificación mixto o para la posible comparación posterior de los resultados obtenidos.

Se pueden identificar en TIE los siguientes bloques lógicos:

- **Filtrado de paquetes.** Se encarga de seleccionar aquellos paquetes que se quieren procesar, haciendo uso de primitivas BPF (*Berkeley Packet Filter*). La forma de expresar este filtrado es mediante la línea de comandos, indicando de forma explícita el filtrado respetando la sintaxis BPF, referenciando un archivo con las expresiones necesarias para el filtrado o mediante comandos de TIE para un filtrado temporal. Los comandos implicados en este proceso son los de la Tabla 2.1.

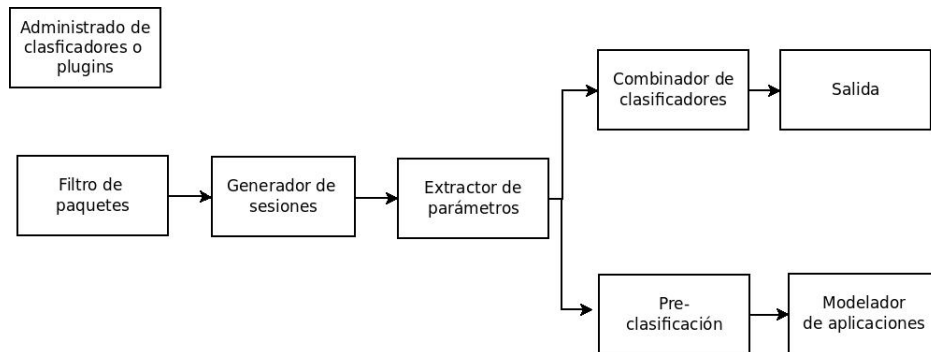


Figura 2.2: Bloques lógicos de TIE [13]

Además, cuenta con un filtrado adicional que no es seleccionable, que se encarga de eliminar aquellos paquetes que puedan generar problemas durante la ejecución de TIE, como son paquetes fragmentados o paquetes con cabeceras IP opcionales, ya que TIE no dispone de las herramientas para procesarlos.

- **Generación de sesiones.** TIE cuenta con complejo sistema para estructurar el almacenado de las diferentes sesiones.

A cada tupla formada por la IP de origen y de destino, por el puerto origen y destino y por el protocolo de la capa de transporte, se le asigna un código *hash* que permite ubicar al flujo dentro de una vector de estructuras. Cada una de estas estructuras almacenará cronológicamente el análisis de todas las sesiones pertenecientes a ese flujo, entre los parámetros almacenados por sesión se encuentran diferentes contadores del número de paquetes procesados, contadores temporales o información acerca del estado de la conexión, como son los *flags*.

- **Extractor de Parámetros.** Este bloque se encarga de extraer diferentes características de los flujos en función de los clasificadores.

TIE, por defecto, extrae los mínimos elementos posibles, como son

Argumentos de TIE de filtrado	
Argumento	Significado
<b>-F</b>	Ruta hacia una archivo de filtrado con primitivas BPF
<b>-C</b>	Los primeros <i>N</i> paquetes no son analizados
<b>-c</b>	Solo se analizan los primeros <i>N</i> paquetes
<b>-D(0-6)</b>	Solo se analizan los paquetes de un día de la semana

Tabla 2.1: Comandos de TIE para el filtrado de paquetes

Argumentos de TIE para la extracción de parámetros	
Argumento	Parámetro que extrae y almacena
<b>-p</b>	Tamaño de la carga útil de los $N$ primeros paquetes
<b>-b</b>	Tamaño de los $N$ primeros paquetes
<b>-P</b>	Carga útil de los primeros $N$ paquetes
<b>-S</b>	Primeros $N$ bytes de una sesión
<b>-I</b>	Intervalo de tiempo para los primeros $N$ paquetes

Tabla 2.2: Comandos de TIE para indicar los parámetros a extraer

aquellos necesarios para controlar el principio y fin de nuevas sesiones, con objeto de aligerar el procesamiento de los diferentes paquetes.

Por ello, cada clasificador ha de indicar el conjunto de parámetros que necesita para llevar a cada su algoritmo de identificación de tráfico durante la ejecución de TIE, es decir, cuando se ejecuta TIE desde el terminal se han de anexionar los argumentos necesarios para almacenar los diferentes parámetros. Algunos de los parámetros disponibles con su respectivo comando asociado se pueden observar en la tabla 2.2.

- Los dos bloques siguientes dependen de si estamos trabajando en modo de clasificación o en modo de entrenamiento.
  - **Combinador de clasificadores.** Este bloque se encarga de combinar los resultados de clasificación de cada uno de los diferentes clasificadores activados. Algunos de estos modos de combinación son: basado en la prioridad de cada clasificador, basado en la máxima probabilidad de asociación<sup>1</sup> o en la decisión mas elegida por el conjunto de los clasificadores.
  - **Salida.** Tramo encargado de imprimir los resultados finales de la clasificación en los diferentes archivos habilitados para ello. Algunos de los comandos de los que TIE dispone cuyo fin esta relacionado están indicados en la tabla 2.3.
  - **Bloque de preclasificación.** Este bloque se encarga de leer el archivo de preclasificación que normalmente se suele denominar *ground\_truth*.

En TIE, este fichero debe de contener un listado de flujos, cada uno de ellos etiquetado con la clasificación a la que pertenece, que servirá para preclasificar los flujos durante la ejecución de TIE, habilitando así la posibilidad de obtener los parámetros necesarios de estos flujos para caracterizar las aplicaciones que los generaron,

<sup>1</sup>Cuando un clasificador atribuye la generación de un flujo a una aplicación, además asocia una probabilidad de la fiabilidad de esa asociación

Argumentos de TIE empleados en la salida de resultados	
Argumento	Significado
<b>-E</b>	Nombre del fichero que almacenará los resultados de la clasificación
<b>-d</b>	Ruta al directorio donde se crearán los ficheros de resultados
<b>-n</b>	Habilita la impresión de los nombres de cada aplicación en los resultados

Tabla 2.3: Argumentos de TIE dedicados a organizar la salida de resultados

para posteriormente ejecutar sobre los parámetros almacenados el algoritmo de entrenamiento de algún clasificador.

- **Bloques de entrenamiento del *plugin*.** Este bloque agrupa al conjunto de funciones cuyo objetivo es almacenar los parámetros necesarios para el entrenamiento, así como a las funciones encargadas de ejecutar el algoritmo de entrenamiento y su posterior impresión en los archivos que almacenen los pertinentes modelos.

Algunos de los argumentos asociados al entrenamiento son los de la Tabla 2.4

Para concluir con los argumentos de línea de comandos de TIE, algunos de ellos, genéricos y necesarios en la ejecución de TIE, vienen expuestos en la Tabla 2.5.

Cabe mencionar que TIE es una plataforma de código abierto, diseñada en C y originalmente dirigida a sistemas operativos tipo Linux, aunque actualmente se ha ampliado a plataformas MacOS así como a FreeBSD.

Básicamente, se trata de un ejecutable donde quedan definidos los diferentes clasificadores disponibles. Adicionalmente, cuenta con una serie de *scripts* que son de utilidad a la hora de comparar los resultados obtenidos por los diferentes clasificadores.

Como se ha mencionado previamente, TIE nos permite desarrollar plantillas de sesiones etiquetadas con la respectiva aplicación a la que pertenece,

Comandos de TIE asociados al entrenamiento de un clasificador	
Comando	Utilidad
<b>-e</b>	Nombre del fichero de referencia para la pre-clasificación
<b>-l</b>	Habilita el entrenamiento, deshabilitando el modo de clasificación
<b>-k</b>	Habilita la clasificación, aún realizándose el entrenamiento

Tabla 2.4: Comandos de TIE relacionados con el entrenamiento de un *plugin*

Argumentos genéricos de TIE	
Argumento	Significado
<b>-a</b>	Indica los <i>plugins</i> que estarán activos en la ejecución de TIE
<b>-r</b>	Establece el archivo o los archivos de los cuales se leerán los tramas

Tabla 2.5: Argumentos generales de TIE

ya que se puede usar el propio fichero de clasificación generado por cualquier clasificador.

Sin embargo actualmente hay pocos *plugins* disponibles en TIE que funcionen correctamente. Algunos de los *plugins* presentes en la versión 1.2 de TIE, que es la más reciente:

- Clasificador basado en puertos, *class\_port*. Este clasificador usa el estándar clásico basado en puertos de la capa de transporte para llevar a cabo la clasificación.
- Clasificador basado en openDPI, *class\_opendpi*. Clasificador que lleva incorporada la última versión de OpenDPI distribuida por Ipoque<sup>2</sup>. Este clasificador aún esta en fase de desarrollo.
- Clasificador basado en layer7, *class\_l7*. Similar a openDPI pero menos aceptado. Implementa el proyecto *l7-filter* que basa la clasificación en realizar una inspección del paquete.

En la página web de TIE podemos ver como hay más *plugins* en desarrollo aunque los que hay incorporados en la versión actual son los mencionados anteriormente. Estos *plugins* en desarrollo usan técnicas muy variadas, como puede ser el estudio del intervalo de tiempo de llegada entre paquetes o en modelos ocultos de markov.

Posteriormente, durante la sección de implementación, se profundizará en los elementos que componen TIE, haciendo especial hincapié en aquellos que tengan una relación directa con el *plugin* que se pretende implementar.

---

<sup>2</sup>Empresa desarrolladora del método openDPI



## Capítulo 3

# Identificación del tráfico en base al tamaño de los mensajes

En este trabajo se pretende implementar un método de clasificación de tráfico basado en el tamaño de los mensajes intercambiados en una sesión, como módulo de TIE.

Este método ha sido propuesto y evaluado recientemente [3] por el equipo de investigación del tutor del TFG. Los resultados obtenidos muestran una buena precisión en la clasificación con un reducido coste computacional, siendo, adicionalmente, un método de clasificación ciega y temprana. Dados estos resultados, resulta altamente interesante el desarrollo de un *plugin* que incorpore esta técnica entre las ya disponibles en TIE, posibilitando así la evaluación y comparación de esta técnica en diversos escenarios.

En este capítulo, se explicarán las características más relevantes del método que se pretende implementar.

### 3.1. Introducción

Tal y como se mencionó en el apartado 2.2, los métodos no invasivos necesitan caracterizar el flujo de una aplicación en función de ciertos elementos identificativos. En el presente trabajo se utilizará el tamaño de los mensajes, en contraposición al habitual uso de las cargas útiles de los paquetes, para caracterizar el intercambio de información asociado a cada aplicación. La motivación para su uso reside en los siguientes aspectos:

1. El análisis de los mensajes resulta, a priori, mas adecuado que el análisis de los paquetes, ya que corresponde a las unidades de información intercambiadas entre las entidades que se comunican y evita los posibles efectos de segmentación presentes en las diferentes redes, lo que

puede ser considerado como una fuente de error.

2. La extracción del tamaño de los mensajes es realmente sencilla, ya que la información necesaria se encuentra presente en la cabecera de la capa de transporte de los paquetes.
3. Uno de los principales puntos a favor del esquema propuesto es que muchas aplicaciones hacen uso de diferentes esquemas conocidos y perfectamente definidos para llevar a cabo sus intercambios de información. De esa forma, las aplicaciones incorporan un conjunto limitado de posibles "métodos", cada uno de los cuales incluye un conjunto específico de mensajes a intercambiar, como por ejemplo HTTP con sus métodos GET, POST o PUT, cada uno de los cuales con una semántica asociadas y unos formatos específicos para los mensajes.

Es relevante indicar que la técnica propuesta no necesita del conocimiento apriori de estos esquemas de comunicación, ni siquiera del número de ellos en cada protocolo, ya que se estimarán los parámetros durante la fase de entrenamiento de los modelos. Esta característica le otorga una ventaja frente a DPI, que necesita la completa la caracterización de aspectos específicos de los mensajes (firmas) para llevar a cabo la inspección de los paquetes.

4. Como se acaba de indicar, cada método tiene asociado un conjunto de mensajes. En algunos casos, el tamaño de estos mensajes es fijo, mientras que en otros casos presentan una distribución que se puede caracterizar estadísticamente.

El método implementado hace uso de distribuciones normales para modelar los tamaños de los mensajes. Estas distribuciones se estiman durante la fase de entrenamiento de forma que cada aplicación tiene asociadas un conjunto de gaussianas características.

5. Muchas aplicaciones, y sus métodos asociados, pueden ser caracterizadas por los esquemas de comunicación que siguen para sus primeros mensajes. Estos mensajes suelen estar relacionados con algún procedimiento de inicialización o establecimiento de la comunicación, como puede ser la autenticación de las partes o el tipo de comunicación a usar.

De acuerdo con esto, el clasificador hace uso esta característica limitando el análisis a los primeros mensajes de la comunicación, permitiendo una identificación temprana de tráfico.



### 3.2. Extracción de características

El primer paso para clasificar un flujo es caracterizarlo adecuadamente en base a un vector de características. En este caso, cada flujo debe ser representado mediante una secuencia de tamaños de los mensajes.

Un mensaje es la verdadera unidad de información transmitida entre las dos entidades de la comunicación, sin tener en cuenta si esta información ha debido ser modificada o dividida en diferentes bloques en función de las características de la red o redes que ha de atravesar.

La diferencia entre tamaño de los paquetes y de los mensajes solo es útil para conexiones TCP, ya que en una conexión UDP cada paquete es un mensaje, al no producirse fragmentación ni retransmisión de los paquetes y tener que adaptar el tamaño de los mismos a la MTU<sup>1</sup> disponible en la red.

De esa forma, cada mensaje es modelado por su tamaño en *bytes* y por un signo. El signo es indicativo de la dirección de la comunicación, que puede ser ascendente(*upstream*) o descendente(*downstream*). En principio se entenderá que un mensaje *upstream* es aquel que ha sido generado por el iniciador de la comunicación, siendo *downstream* si tiene lugar en la dirección contraria. Dado que en algunos casos no es sencillo determinar el iniciador de la comunicación, se considera como tal la entidad correspondiente a la dirección de origen del primer paquete observado.

En una comunicación TCP, para contabilizar el tamaño de cada mensaje es necesario la inspección de la cabecera de la capa de transporte, con objeto de detectar cualquier secuencia de paquetes que en sí misma un paquete. Este procedimiento permite superar los siguientes dos problemas que son:

- Los mensajes ACK usados en TCP son útiles a la hora de verificar que un paquete a llegado a su destino. Sin embargo, pueden ser una fuente de ruido a la hora de determinar si un mensaje a terminado o no en función de la implementación del clasificador.

En nuestro método los paquetes ACK puros son ignorados consiguiéndose así que no sean delimitadores del tamaño del mensaje.

- Algunas aplicaciones realizan intercambios de información en los que el tamaño del mensaje que se pretende transmitir es mayor que el MTU de la red, lo cual provoca la fragmentación de la información.

Es aquí donde entra el método propuesta, ya que evita este problema y aúna todos los paquetes que pertenecen a un mismo mensaje, obteniéndose así el tamaño real de la información que se pretendía transmitir.

---

<sup>1</sup>MTU ( *Maximum Transfer Unit*): hace referencia al valor máximo de tamaño de paquete que se puede enviar en una red.

El número de mensajes considerado en el vector de parámetros,  $L$ , es predefinido globalmente por el sistema, en este caso desde TIE. Idealmente,  $L$  debe de ser lo más pequeña que se pueda para conseguir una identificación temprana y para que se puedan clasificar flujos de pocos mensajes.

Dado un flujo,  $F_i(a \rightarrow b)$ , iniciado por el usuario  $a$  y dirigido hacia  $b$ , se creará un vector,  $O_i$ , de  $L$  enteros con signo, correspondientes al tamaño de mensaje y a sus respectivas direcciones. Como se ha indicado anteriormente, los paquetes de acuerdo de TCP y los paquetes puros ACK no serán tenidos en cuenta durante la estimación de este vector. Esto es:

$$O_i = \{s_1, s_2, \dots, s_L\} \begin{cases} \text{tamaño}\{\text{mensaje}_i\} & \text{si } \text{mensaje}_i(a \rightarrow b) \\ -\text{tamaño}\{\text{mensaje}_i\} & \text{si } \text{mensaje}_i(b \rightarrow a) \end{cases} \quad (3.1)$$

### 3.2.1. Delimitación del mensaje

Con respecto a el procedimiento de detección de estos mensajes, ha de realizarse una distinción entre UDP y TCP. Obviamente, en los flujos UDP la carga útil de cada paquete corresponde con un mensaje y no debe realizarse ninguna distinción. Sin embargo, para TCP se tomará que las cargas útiles pertenecientes a paquetes enviados consecutivamente en la misma dirección de la comunicación, se considerarán como un solo mensaje si no se presenta alguno de los siguientes elementos delimitadores:

- El siguiente paquete con contenido<sup>2</sup> tiene la dirección contraria. Es evidente que si una entidad termina y comienza la otra entidad a responder es porque la primera ha terminado de enviar la información del mensaje actual.
- Si el paquete contiene el flag PUSH de TCP activo. Este flag es normalmente indicativo de que el emisor ha terminado de enviar la información perteneciente al mensaje actual.
- El tamaño de la carga útil es inferior al MSS<sup>3</sup>. Este valor se determina durante el *handshake* de TCP. La justificación de este caso reside en la hipótesis de que las transferencias de mensajes grandes se realizan mediante paquetes del mayor tamaño posible.

Consecuentemente, se establece un valor mínimo a partir del cual se determinará si un paquete es demasiado pequeño como para suponer que el siguiente paquete puede pertenecer al mismo mensaje.

Este limitador es especialmente útil para ciertas aplicaciones de chat (como MSN) o de control remoto (como SSH o TELNET) en las que se pueden enviar varios mensajes en la misma dirección.

---

<sup>2</sup>No un paquete ACK puro

<sup>3</sup>MSS (*Maximum Segment Size*) es el tamaño más grande del segmento de datos que se puede utilizar durante la comunicación

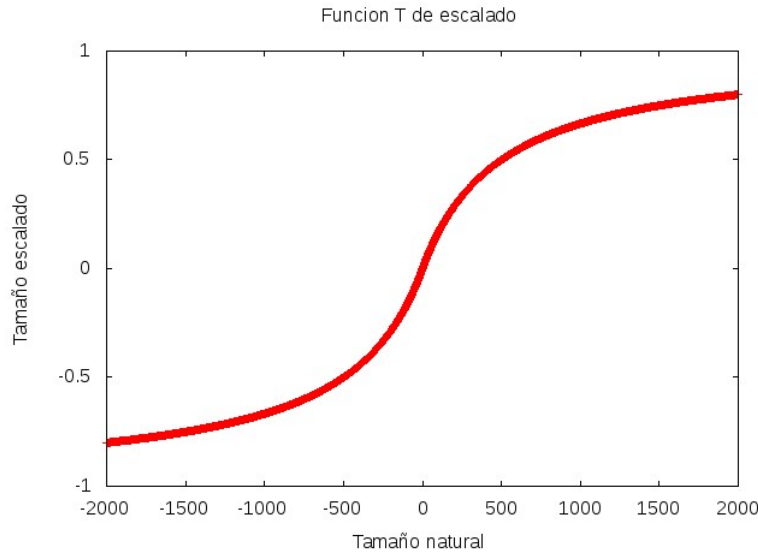


Figura 3.1: Funcion de escalado

### 3.2.2. Normalización del tamaño de los mensajes

Una vez descrito cómo parametrizar un flujo, resulta relevante resaltar que lo que este método pretende es diferenciar los mensajes en función de sus tamaños.

En este entido, es sencillo comprender que no será de igual relevante la diferencia de 100 *bytes* entre un mensaje de 2 *bytes* y otro de 102 *bytes* que entre dos mensajes de 1200 y de 1300 *bytes*, siendo mucho mas significativa la primera al ser los tamaños de los mensajes mucho menores.

Es aquí donde nace la necesidad de definir una distancia que pondere adecuadamente estas diferencias de acuerdo a los objetivos del método.

Así, en [3] se propone una transformación  $T$  que lleva a todo tamaño de mensaje a un espacio normalizado en un rango de valores de  $] -1, 1[$  que nos permite escalar el tamaño de cualquier mensaje. La transformación  $T$ , denominanda en lo sucesivo función de escalad, queda definida como:

$$T(s) : \mathbb{Z}^* \rightarrow ] -1, 1[ \quad s \rightarrow s' = \frac{s}{(B + |s|)} \quad (3.2)$$

donde  $B$  es un valor que servirá para diferencia entre mensajes considerados grandes y pequeños.

En la Figura 3.1 se muestra la transformación propuesta en función del valor el tamaño del mensaje. Se puede comprobar como conforme aumenta el tamaño original del mensaje va disminuyendo la repercusión de dicho aumento sobre el tamaño escalado.

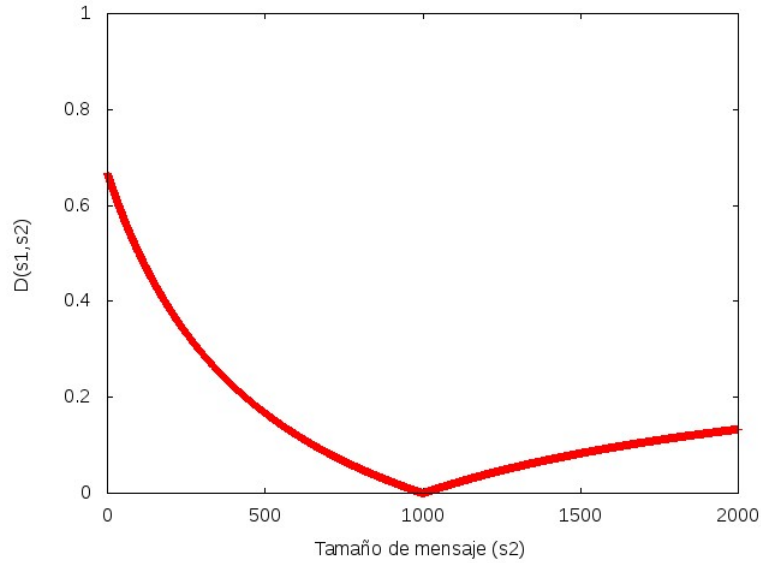


Figura 3.2: Distancia escalada

Utilizando esta función de escalado, se define una distancia para evaluar las diferencias de tamaño según:

$$D(s_1, s_2) = \begin{cases} 1 & \text{si } \text{sgn}(s_1) \neq \text{sgn}(s_2) \\ \left| \frac{s_1}{(B + |s_1|)} - \frac{s_2}{(B + |s_2|)} \right| & \text{otro caso} \end{cases} \quad (3.3)$$

Esta distancia simplemente aplica la distancia euclidiana a los tamaños de mensaje escalados si son de la misma dirección. En caso de que la dirección entre dos mensajes sea distinta, la distancia será 1, que es el valor máximo, lo que es coherente con la finalidad pretendida, al ser esta la máxima distancia para mensajes en dirección contrario, sean cuales sean sus tamaños de mensaje.

En la Figura 3.2 se muestra gráficamente la evolución de esta distancia fijando el valor del tamaño de uno de los mensajes  $s_1$  en 1000 bytes y considerando un tamaño  $s_2$  variable.

Se observa como, efectivamente, es mucho mayor la distancia para un mensaje de un tamaño pequeño que para un mensaje mayor, aún estando separados por la misma diferencia de *bytes* con respecto al tamaño de mensaje de referencia.

### 3.3. Sistema de clasificación

El método propuesto en [3] para la identificación de tráfico se basa en la utilización de un clasificador compuesto de modelos de Markov, cada uno

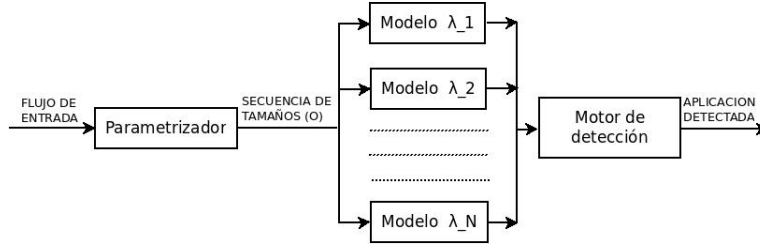


Figura 3.3: Módulos del clasificador

de los cuales corresponde a una de las aplicaciones a identificar. De esta forma, el sistema de clasificación está compuesto por 3 módulos o bloques principales (Figura 3.3):

- Un módulo de parametrización, que obtiene el vector  $O_i$  de tamaños de mensajes,  $s_l$ , para cada uno de los flujos  $F_i$

$$O_i = \{s_1, s_2 \dots s_L\} \quad (3.4)$$

siendo  $L$  el número de mensajes iniciales considerados para llevar a cabo la clasificación.

- Un módulo, compuesto por un conjunto de  $N$  modelos de Markov que evalúa las probabilidades de los vectores de tamaños de mensajes para todas las aplicaciones consideradas:

$$P(O_i|\lambda_n) \quad 1 \leq n \leq N \quad (3.5)$$

donde  $N$  es el número total de aplicaciones identificables.

- El motor detector, que se limitará a tomar aquella aplicación para la que la probabilidad de generar el vector de mensajes,  $O_i$  sea máxima

$$clasifica(O_i) \rightarrow \operatorname{argmax}_n \{P(O_i|\lambda_n) \quad 1 \leq n \leq N\} \quad (3.6)$$

el valor de  $P(O_i|\lambda_n)$  corresponde a la probabilidad de que, dada una aplicación esta haya generado el conjunto de observables,  $O_i$ .

Sin embargo, la identificación debería realizarse a partir de la probabilidad de la aplicación  $n$ , dada la secuencia de observables  $O_i$ , esto es, a partir de:

$$clasifica(O_i) \rightarrow \operatorname{argmax}_n \{P(\lambda_n|O_i) \quad 1 \leq n \leq N\} \quad (3.7)$$

Aplicando el teorema de Bayes se puede expresar  $P(\lambda_n|O_i)$  de la siguiente forma:

$$P(\lambda_n|O_i) = \frac{P(\lambda_n)P(O_i|\lambda_n)}{P(O_i)} \quad (3.8)$$

Como la probabilidad de una secuencia de observaciones  $P(O_i)$  es independiente del modelo, este factor se puede eliminar de la evaluación ya que no afecta al criterio de decisión por ser igual para todos los casos. Asumiendo además que todos los modelos son equiprobables, se puede utilizar directamente la expresión 3.6.

En cualquier caso, en realidad no todas las aplicaciones son igual de probables, ya que un flujo de HTML es mucho más común que no NetBIOS o que uno de ICMP. Esto podría ser útil a la hora de caracterizar el tráfico de una red, mediante un estudio de frecuencia de observación de tipos flujos.

Sin embargo al asumir que  $P(\lambda_n)$  se busca que el modelo no este basado en componenets dinámicos de la red tales como:

- La probabilidad de las diferentes aplicaciones  $P(\lambda_n)$  no es estacionaria, ya que dependiendo del uso que hagan los usuarios de la red esta podrá ir combinando diferentes tipos de aplicaciones mayoritarias a lo largo del tiempo.
- Esta probabilidad depende del entorno de la red, es decir, este modelado puede ser útil en una red previamente caracterizada, pero si empieza a ser aplicado en otra red, este modelado ya no tendrá ninguna validez.
- El hecho de tener que caracterizar la tipología de tráfico de una red complicaría en exceso la labor del clasificador, siendo la sencillez y la rapidez de clasificación las principales características del método que se pretende implementar.

Queda por definir esta probabilidad  $P(O_i|\lambda_n)$  que viene dada por el orden de los observables,  $O_i$ , produciendo una cadena de Markov.

### 3.3.1. Modelado mediante gaussianas múltiplo

Como se ha indicado, el sistema propuesto asocia cada aplicación a un modelo de Markov de izquierda-a-derecha, donde cada mensaje es tratado como un estado de dicha cadena: el mensaje con el índice 1 se corresponde con el estado 1, el mensaje con el índice 2 se corresponde con el estado 2, y así sucesivamente.

El conjunto compuesto por los  $N$  modelos, uno por cada aplicación, es estimado durante el entrenamiento del sistema:

$$\Lambda = \{\lambda_n \quad 1 \leq n \leq N\} \quad (3.9)$$

como parte de este entrenamiento, un conjunto de gaussianas para cada conjunto de observaciones de una aplicación  $n$ , es generado,  $G$ . Con diferente modelo para cada estado de esa aplicación  $n$ :

$$G = \{g(n, l) \quad 1 \leq n \leq N, 1 \leq l \leq L\} \quad (3.10)$$

por otro lado, el vector de observaciones,  $O_i$ , ha de ser evaluado por el sistema, de acuerdo con los diferentes modelos para cada una de las aplicaciones 3.5.

Como se ha indicado, al tratarse de un modelo de Markov de izquierda-derecha, solo han de evaluarse la probabilidades de observación. Esto es, la probabilidad de que las observaciones,  $O_i$ , hayan sido generadas por una aplicación  $n$  puede ser calculada como el producto de la probabilidad de observación para los diferentes estados de la cadena:

$$P(O_i|\lambda_n) = \prod_{l=1}^L P(s_i|g(n, l)) \quad (3.11)$$

el modelado de  $g(n, l)$  realizado durante la fase de entrenamiento mediante agrupamiento o *clustering*, estima un conjunto de gaussianas multiplico. Cada uno de estas gaussianas vendrá caracterizado por su *centroide*, peso relativo y varianza de muestras.

El motivo de esta aproximación es que, tal y como mencionó en el apartado 3.1, muchas aplicaciones usan esquemas de comunicación predeterminados en los cuales hay un número finito de métodos posibles. Las gaussianas multiplico responden a esta característica, permitiendo disponer de diferentes puntos de alta probabilidad, que se han de ajustar a los tamaños de mensaje que cada aplicación genera para sus diversos métodos.

Por otra parte, la influencia de cada gaussiana individual que se estime en la distribución del modelo final debe ser pesada teniendo mayor relevancia aquellas que se ajusten a los métodos que son mas comunes en la aplicación. Cómo se detallará más adelante, se utilizarán algoritmos de agrupamiento para estimar cada una de estas distribuciones de forma no supervisada, esto es, sin necesidad de conocer cuáles de las muestras corresponden a cada método ni el número de métodos existentes para cada aplicación.

En consecuencia, la probabilidad de que un mensaje en la posición  $l$  de la secuencia de observables,  $O_i$ , haya sido generado por una aplicación  $n$  vendrá determinada por este conjunto de gaussianas, donde cada una de ellas responderá a cada uno de los diferentes métodos inherentes a cada aplicación, aunque no sean conocidos apriori.

Así, para cada estado del modelo,  $l$ , asociado a la aplicación,  $n$ , se define una gaussiana multiplico compuesta por  $K_{l,n}$  gaussianas dentro del modelo  $g(n, l)$  de forma que la probabilidad de que un tamaño de mensaje,  $s_l$ , corresponda a la distribución  $c_{i,l,n}$  viene dada por:

$$P_{C_{i,n,l}}(s_l) = \frac{1}{\sigma_{C_{i,n,l}}\sqrt{2\pi}} * e^{-0.5 \left[ \frac{D(s_l, \mu_{C_{i,n,l}})}{\sigma_{C_{i,n,l}}} \right]^2} \quad (3.12)$$

donde la media,  $\mu_{C_{i,n,l}}$ , y la varianza,  $\sigma_{C_{i,n,l}}$ , serán estimados durante el entrenamiento del modelo. Además la función de distancia,  $D(s_l, \mu_{C_{i,n,l}})$ , es la definida en el Apartado 3.2.2.

Partiendo de la expresión anterior, se tiene que la probabilidad de que una observación,  $s_l$ , haya sido generada por la aplicación  $n$  en la posición  $l$  viene dada por:

$$P(s_l|g(n, l)) = \sum_{i=1}^{K_{n,l}} w_{C_{i,n,l}} * P_{C_{i,n,l}}(s_l) \quad (3.13)$$

En esta ecuación la componente  $w_{C_{i,n,l}}$  es el peso de cada gaussiana presente en el modelo  $g(n, l)$  y  $K_{n,l}$ , es el número de picos de la gaussiana, obtenido durante la fase de entrenamiento. Cabe mencionar que deberán existir dos distribuciones para cada posición de la secuencia de cada aplicación, uno para los observables *upstream* y otro para los observables *downstream*, convirtiendo a la dirección del mensaje en el factor más relevante de la clasificación.

Con este aislamiento basado en las direcciones se pretende distinguir entre los métodos dirigidos del cliente al servidor de los que siguen el camino contrario, evitando que los modelos de gaussianas definidos para una dirección sean también aplicables para la otra dirección.

### 3.4. Entrenamiento del modelo

El entrenamiento del modelo deberá determinar, básicamente, las distribuciones de probabilidad de las observaciones, ya que tanto la topología, como el número de estados de los modelos de Markov quedan fijados por el sistema. Por tanto, es necesario estimar los modelos  $g(n, l)$  para todas las aplicaciones y para cada uno de los diferentes estados, para lo que se utilizan flujos convenientemente etiquetados según la clase a la que correspondan (valores  $n$  y  $l$ ). Sin embargo, no se dispone de información correspondiente a los métodos asociados a cada aplicación y/o estado, por lo que el número de picos finales que corresponden a las distintas gaussianas es desconocido.

En consecuencia, el entrenamiento debe ser no supervisado en relación a estos factores, para lo que se propone un algoritmo de *clustering* para agrupar las observaciones en diversos conjuntos supuestamente asociados cada uno a un método.

Este tipo de procedimientos son útiles a la hora de organizar un conjunto de datos y agruparlos en torno a unos *clusters* representativos del espacio de datos, siendo el centro de un *clúster*<sup>4</sup> lo que se conoce como *centroide*.

Los algoritmos de *clustering* pueden ser divididos en dos tipos [14]:

- **Algoritmo de *clustering* jerárquico:** Al comenzar el algoritmo se le asigna a cada dato su propio *clúster* y se van uniando entre sí aquellos

---

<sup>4</sup>Aunque el término *clúster* no está recogido en el diccionario de la RAE, sí que es un término ampliamente utilizado siempre que se trata el ámbito de los algoritmos de agrupamiento, por ello a lo largo de esta trabajo se empleará *clúster* para una agrupación y *clusters* para un conjunto de agrupaciones



*clusters* que son muy similares, reduciéndose así el número de *clusters* a la vez que se consigue una estructura jerárquica en función de aquellos que son más representativos, es decir, en la cúspide de la jerarquía estarán aquellos *clusters* que agrupen más datos.

- **Algoritmo de *clustering* basado en particiones:** Estos algoritmos ven cada *clúster* como un modelo representativo del conjunto de datos sin imponer una estructura jerárquica. Uno de los algoritmos basados en particiones más usados es el denominado K-medias (*k-means*) [15], que será el que se implementará en este trabajo.

Como se ha indicado, para este procedimiento es necesario disponer de un conjunto de observables etiquetados con la aplicación a la que pertenecen, siendo examinados con el objetivo de obtener la secuencia de mensajes presentes en los observables, utilizando para ello el conjunto de delimitadores de mensajes explicados en el Apartado 3.2.1.

Se agruparán las muestras de mensajes para *upstream* y para *downstream* ya que los *clusters* serán diferentes para ambas direcciones. Posteriormente se aplica el algoritmo de *K-medias* con objeto de centrar los *clusters* en torno a los tamaños de mensajes mas característicos de una aplicación  $n$  para la posición  $l$  y para la dirección correspondiente, permitiéndose así obtener los métodos, definidos por el tamaño de sus mensajes generados, característicos de esa aplicación aunque puedan ser previamente desconocidos.

### 3.4.1. Algoritmo *K-medias*

Los principales elementos de *K-medias* son los siguientes:

- Sea  $X = \{x_i, i = 1, \dots, N\}$  el conjunto de  $n$  datos de entrada al algoritmo.
- Sea  $C = \{c_k, k = 1, \dots, K\}$  el conjunto de  $K$  *clusters*.
- Sea  $\zeta = \{\zeta_k, k = 1, \dots, K\}$  el conjunto de los  $K$  *centroides* de los *clusters*.

El algoritmo de agrupamiento debe asociar cada uno de los elementos de  $X$  a uno de los *clusters* existente ( $C$ ). En el caso de K-medias, el objetivo del algoritmo es reducir el error cuadrático medio sobre todo el conjunto de datos. Normalmente, se suele utilizar como medio de cálculo de este error para cada punto la distancia entre un punto,  $x_i$ , y el centroide,  $\zeta_k$ , del grupo al que se asocia ese punto,  $x_i \in C_k$ . El error total asociado a un *clúster* de acuerdo a esta definición viene dado por:

$$\xi(c_k) = \sum_{x_i \in c_k} \|x_i - \zeta_k\|^2 \quad (3.14)$$

en esta expresión anterior el valor de  $\zeta_k$  se puede sustituir por  $\mu_k$  en el caso de usarse la distancia euclidiana, ya que es la media el *centroide* que hace mínimo el error. Sin embargo, en el modelo que se pretende implementar se está usando un espacio escalado donde la definición de error será la distancia escalada (eq. 3.3). Por lo tanto la expresión anterior quedaría:

$$\xi(c_k) = \sum_{x_i \in c_k} |D(x_i, \zeta_k)|^2 \quad (3.15)$$

siendo el error final a minimizar la suma de los errores producidos en todos los *clusters*

$$\xi(C) = \sum_{k=1}^K \xi(c_k) = \sum_{k=1}^K \sum_{x_i \in c_k} |D(x_i, \zeta_k)|^2 \quad (3.16)$$

Es fácil intuir, que conforme se aumente el número de *clusters*,  $K$ , se irá reduciendo el error cuadrático,  $\xi(C)$ . No obstante, el objetivo del mecanismo es encontrar el mínimo número de *clusters* que reduzcan lo máximo posible ese error, para lo que se utilizan umbrales de error.

El algoritmo de K-medias básico se inicializa con un grupo de  $K$  *clusters*, cada uno con sus muestras preasignadas por algún procedimiento, y se reasignan las muestras a estos *clusters* para ir reduciendo el error, teniéndose siempre un número fijo de *clusters*. Los principales pasos son los siguientes:

1. Se seleccionan una partición inicial de  $K$  *clusters* con sus *centroides* preasignados.
2. Se reestima la pertenencia de cada muestra a cada *clúster*, es decir, al *centroide* más cercano.
3. Se calculan los nuevos *centroides*.
4. Tras esto, se irán repitiendo los pasos 2 y 3 hasta que se consiga que los miembros de cada *clúster* se establezcan.

Este algoritmo básico ha sido objeto de muchas variaciones a lo largo de la historia, siendo numerosas las variantes cada una con ciertas ventajas e inconvenientes frente a otras en función de los datos y el problema abordado. En nuestro caso se implementará un algoritmo denominado K-medias iterativo cuyas principales características o diferencias con respecto al algoritmo básico antes expuesto son las siguientes:

- Al comenzar el algoritmo se considerará únicamente un *clúster* que se irá dividiendo hasta llegar al número de grupos requerido. Aquí se pueden tener en cuenta diferentes metodologías para saber si se ha llegado al número ideal de *clusters*. Caben dos posibilidades básicas:

- Forzar que se llegue a un número fijo de *clusters*. Esta especificación puede provocar que algunos *clusters* queden vacíos o que el aumento del número *clusters* no tenga suficiente importancia en la reducción del error cuadrático medio,  $\xi(C)$ .
- Limitar el aumento de *clusters* en función de la reducción del error  $\xi(C)$  conseguida. Aquí se debería fijar un umbral de reducción de error a partir del cual no se deba continuar con ese mecanismo de seguir dividiendo los clusteres.

En la implementación realizada finalmente se ha optado por una solución mixta que limita el número *clusters* pero que capacite al algoritmo para caracterizar los métodos presentes en las aplicaciones y, además, se establece un umbral para evitar un aumento de *clusters* inútil con respecto a la disminución del error.

- Otro punto discordante con respecto al modelo clásico es la forma de dividir los clusteres. Se plantean dos formas distintas:

- Se dividirá en cada iteración solo uno de los clusteres. El *clúster* que se partirá será aquel que tenga el mayor error cuadrático,  $\xi(c_k)$ .
- En cada iteración se divide cada uno de los *clusters* presentes en otros dos. Es decir, la secuencia del número de clusteres sería 1, 2, 4, 8, 16, ... siguiendo una progresión cuadrática.

Para este caso se deberá seguir esta progresión hasta que el número de *clusters* sea el máximo potencia de dos que siga siendo menor que el  $K$  máximo fijado. Se continuará con el otro modelo hasta que se llegue a  $K$ .

En el modelo implementado ambas opciones pueden ser seleccionables.

Expuestas las modificaciones realizadas hay que aclarar que el modelo clásico se seguirá usando, ya que hace falta recalcular los *centroides* cada vez que se aumenta el número de clusteres en el modelado. Cabe mencionar que la forma de calcular de este *centroide* también deberá tener en cuenta que se está trabajando en un espacio escalado, por lo que este cálculo no será inmediato.

### 3.4.2. Cálculo de *centroides*

Los pasos a seguir para calcular de forma correcta el centroide  $\zeta_k$  de cualquier *clúster*  $C_k$  son:

1. En primer lugar, se deberán trasladar todas las muestras  $x_i \in C_k$  al espacio escalado aplicando la expresión 3.2, dando como resultado,  $\chi_i$ ,

un valor comprendido entre  $]0, 1[$ , para mensajes *upstream*, o  $] - 1, 0[$ , para mensajes *downstream*. Por tanto se tiene:

$$\chi_i = \frac{x_i}{(B + |x_i|)} \quad (3.17)$$

2. Se calculará la media aritmética de estos tamaños de mensajes escalados,  $\vartheta$ :

$$\vartheta_k = \frac{1}{N_k} \sum_{\chi_i \in C_k} \chi_i \quad (3.18)$$

donde  $N_k$  es el número de muestras contenidas en  $C_k$ .

3. Se realiza la operación inversa<sup>5</sup> al paso 1 devolviendo al *centroide* su tamaño representación original,  $\zeta_k$ . En este caso deberemos de realizar dos operaciones distintas debido al valor absoluto de la expresión 3.17.

Para *upstream* (signo positivo):

$$\zeta_k = \frac{B * \vartheta_k}{1 - \vartheta_k} \quad (3.19)$$

Para *downstream* (signo negativo):

$$\zeta_k = \frac{B * \vartheta_k}{1 + \vartheta_k} \quad (3.20)$$

Si se observa la Figura 3.1, se ve cómo esta es simétrica con respecto a un mensaje de tamaño 0 (eje Y). Por tanto, el procedimiento de cálculo del *centroide* proporciona el mismo resultado, en valor absoluto, para *downstream* que para *upstream* si las muestras tienen el mismo valor absoluto.

Lo que se está planteando es que el mecanismo anterior es igual para ambas direcciones si se toma el valor absoluto de las muestras de entrada  $x_i$  y, posteriormente, se cambia el signo del resultado en función de si las muestras eran *downstream* o *upstream*. Los pasos para este otro procedimiento común para ambas direcciones son:

1. Se toma el valor absoluto de las muestras

$$x_{i,abs} = |x_i| \quad (3.21)$$

2. Se escala este valor absoluto:

$$\chi_{i,abs} = \frac{x_{i,abs}}{(B + |x_{i,abs}|)} \quad (3.22)$$

---

<sup>5</sup>En la parte de futuro trabajo se comentará una forma de evitar tener que estas pasando constantemente entre los distintos espacios

3. Se calcula la media aritmética para estas muestras en valor absoluto ya escaladas:

$$\vartheta_{abs} = \frac{1}{N_k} \sum_{\chi_{i,abs} \in C_k} \chi_{i,abs} \quad (3.23)$$

4. Se realiza el de-escalado del resultado anterior:

$$\zeta_{k,abs} = \frac{B * \vartheta_{k,abs}}{1 - \vartheta_{k,abs}} \quad (3.24)$$

5. Se coloca el signo en función de la dirección del mensaje:

$$\begin{cases} \zeta_k = +\zeta_{k,abs} & dir : upstream \\ \zeta_k = -\zeta_{k,abs} & dir : downstream \end{cases} \quad (3.25)$$

Expuesto que se puede unificar el proceso de hallar los *centroides*, aún queda por definir el modelado de la bisección de los *clusters*, medio usado para ir aumentando progresivamente el número de *clusters* en el modelado. De nuevo, el hecho de no estar trabajando en un espacio euclídeo tendrá un papel fundamental.

### 3.4.3. Bisección de clústeres

La bipartición de un *clúster* se realiza a partir de considerar dos nuevos *centroides* mediante el desplazamiento, en direcciones opuestas y en una distancia aleatoria pequeña, del *centroide* original del *clúster*. Cada uno de estos nuevos *centroides* define un *clúster* diferente al que se asignan los elementos originales en el *clúster* inicial, de acuerdo a la distancia a los dos nuevos *centroides*.

Antes se han mencionado las dos formas de llevar a cabo esta bisección; globalmente a todos los *clusters* o solamente al *clúster* más disperso. En caso de que sea al más disperso, se tomará aquel *clúster* que presente la mayor varianza, aunque de nuevo la varianza se calculará de diferente forma a la habitual para adecuarla al espacio en el que se trabaja:

$$\sigma_i^2 = \frac{1}{N} \sum_{j=1}^N \left( \frac{\zeta_i}{B + |\zeta_k|} - \frac{x_i}{M + |x_i|} \right)^2 \quad (3.26)$$

O, expresando las variables en el dominio escalado:

$$\sigma_i^2 = \frac{1}{N} \sum_{j=1}^{N_i} (\zeta_i - \chi_i)^2. \quad (3.27)$$

A continuación se explicará el proceso para llevar a cabo esta bisección sobre un *clúster* dado,  $\zeta_k$ , que dará como resultado un *clúster*  $\zeta_{k,+}$  y otro

$\zeta_{k,-}$  tales que, su valor absoluto, debe ser mayor y menor que el del *centroide* del *clúster* original,

$$\begin{cases} |\zeta_{k,+}| > |\zeta_k| \\ |\zeta_{k,-}| < |\zeta_k| \end{cases} \quad (3.28)$$

De nuevo se tiene que el procedimiento es similar para *downstream* que para *upstream* debido a la naturaleza simétrica de la función de escalado. Por ello, se expondrá el procedimiento que se ha de realizar para *upstream* siendo el de *downstream* equivalente. Únicamente hay que utilizar el valor absoluto, consiguiéndose tratar las muestras como positivas y, cuando concluya el proceso, cambiar el signo al resultado, es decir, a los dos *centroides*  $\zeta_{K,+}$  y  $\zeta_{K,-}$ .

Tomados ya en valor absoluto, se tendrá que estos *centroides* vendrán definidos por la propia desviación típica,  $\sigma$ , que será multiplicada por,  $\eta$ , para modular el desplazamiento del *centroide*

$$\begin{cases} |\zeta_{k,+}| > |\zeta_k| & \eta\sigma = \left( \frac{|\zeta_{k,+}|}{B + |\zeta_{k,+}|} - \frac{|\zeta_k|}{B + |\zeta_k|} \right) \\ |\zeta_{k,-}| < |\zeta_k| & \eta\sigma = \left( \frac{|\zeta_k|}{B + |\zeta_k|} - \frac{|\zeta_{k,-}|}{B + |\zeta_{k,-}|} \right) \end{cases} \quad (3.29)$$

Eliminando los denominadores:

$$\begin{cases} |\zeta_{k,+}| > |\zeta_k| & \eta\sigma(B + |\zeta_k|)(M + |\zeta_{k,+}|) = (|\zeta_{k,+}|)(B + |\zeta_k|) - (|\zeta_k|)(B + |\zeta_{k,+}|) \\ |\zeta_{k,-}| < |\zeta_k| & \eta\sigma(B + |\zeta_k|)(M + |\zeta_{k,-}|) = (|\zeta_k|)(B + |\zeta_{k,-}|) - (|\zeta_{k,-}|)(B + |\zeta_k|) \end{cases} \quad (3.30)$$

Y, finalmente, despejando la expresión para cada *centroide*:

$$\begin{cases} |\zeta_{k,+}| > |\zeta_k| & |C_{\zeta,+}| = \frac{|\zeta_k| + \eta\sigma(B + |\zeta_k|)}{1 - \sigma\eta(1 + \frac{|\zeta_k|}{B})} \\ |\zeta_{k,-}| < |\zeta_k| & |C_{\zeta,-}| = \frac{|\zeta_k| - \eta\sigma(B + |\zeta_k|)}{1 + \sigma\eta(1 + \frac{|\zeta_k|}{B})} \end{cases} \quad 0 < \eta < 1 \quad (3.31)$$

De nuevo, se debe indicar el signo de los nuevos *centroides*, en función de la dirección del mensaje:

1. Para *upstream*:

$$\begin{cases} \zeta_{k,+} = +|\zeta_{k,+}| & \text{dir : upstream} \\ \zeta_{k,-} = +|\zeta_{k,-}| & \text{dir : upstream} \end{cases} \quad (3.32)$$

2. Para *downstream*

$$\begin{cases} \zeta_{k,+} = -|\zeta_{k,+}| & \text{dir : downstream} \\ \zeta_{k,-} = -|\zeta_{k,-}| & \text{dir : downstream} \end{cases} \quad (3.33)$$

#### 3.4.4. Pasos del modelo

Resumiendo, el algoritmo de K-medias modificado a implementar pretende reducir el error cuadrático medio total,  $\xi(C)$ , siguiendo estos pasos:

1. Previamente al comienzo del algoritmo, se fija un número máximo de  $K$  *clusters*, así como un umbral de recuperación de error.
2. Se parte de un único *clúster* que aglutinará a todas las muestras.
3. Se inicia el proceso de bisección de clusteres. Hay dos opciones:
  - Se divide el *clúster* más variado.
  - Se dividen todos si el doble del número de *clusters* actuales es menor que el número de *clusters* máximo fijado.
4. Se reagrupan las muestras en función de su proximidad con respecto a los *centroides* del conjunto de *clusters*.
5. Se estiman los *centroides* y la varianza para los nuevos *clusters*.
6. Se realiza el procedimiento de K-medias básico con objeto de reducir al máximo el error para el número actual de *clusters*.
7. Se itera a partir del paso 3 hasta que no se consiga reducir el error de forma significativa o se llegue al máximo número de *clusters* posible.





## Capítulo 4

# Implementación práctica

En este capítulo se explicará el trabajo realizado para implementar el método expuesto en el capítulo anterior y hacerlo compatible con TIE, siendo lo segundo el principal objetivo.

Para ello se comenzará analizando cómo TIE consigue comunicarse con el conjunto de clasificadores, predefiniendo para ello una estructura que cualquier desarrollador ha de respetar a la hora de implementar un *plugin*

Se continuará detallando como se ha construido un extractor de parámetros de cada flujo diferente al estándar en TIE, lo que resulta de especial interés ya que, en principio, este bloque en TIE no registra los elementos que son necesarios para la parametrización necesaria para el método a implementar.

Finalmente, se comenta la implementación del identificador de tráfico, continuando con la sección del entrenador que se encarga de crear los diferentes clústeres basándose en el algoritmo de K-medias.

### 4.1. Arquitectura del clasificador

Para crear un clasificador compatible con TIE se debe partir del esqueleto de *plugin* establecido por los creadores de esta plataforma. Este no es más que el conjunto de funciones a las cuales TIE se dirige desde su código primario para comunicarse con sus *plugins*.

De acuerdo con las especificaciones de TIE, estas funciones deben estar contenidas en un archivo que se denomina *class.xxx.c*, siendo *xxx* el nombre del clasificador. Para este clasificador, el nombre elegido ha sido MSBC (*Message Size Based Classifier*).

El código fuente generado en C deberá estar ubicado junto al código fuente de TIE, dentro de una carpeta llamada *plugins* y a su vez dentro de otra carpeta con el nombre del clasificador. Ubicar el clasificador aquí permite que TIE, durante la fase de compilación, consiga localizar al clasificador, asociando su código al de TIE, permitiendo su posterior activación durante la fase de ejecución. TIE hace uso de punteros dinámicos, cargando en cada

ejecución los clasificadores que se hayan solicitado.

Las funciones necesarias en el *fichero class\_MSBC.c* se pueden agrupar en funciones de activación, comunes al modo de entrenamiento y al de clasificación, y los dos bloques de funciones correspondientes a esos dos modos:

- Funciones de activación:
  - ***p\_enable***: Comprueba que se puede habilitar el clasificador, para ello se analiza si se han pasado por el terminal, en la orden de ejecución de TIE, los parámetros necesitados por el clasificador.
  - ***p\_disable***: Deshabilita el clasificador. Esta función es llamada si no se cumple la función *p\_enable*.
  - ***p\_load\_signatures***: Carga los modelos de las distintas aplicaciones, necesarios para la clasificación de sesiones.
- Funciones de clasificación:
  - ***p\_is\_session\_classifiable***. Función encargada de evaluar si se han recogido los parámetros necesarios de una sesión para poder ser clasificada.
  - ***p\_classify\_session***: Es el motor de clasificación. Analiza las características almacenadas de cada sesión, atribuyendo a esa sesión la aplicación que la generó.
  - ***p\_dump\_statistics***: Imprime datos de la clasificación realizada.
- Funciones de entrenamiento:
  - ***p\_session\_sign***: Esta función se encarga de almacenar aquellos parámetros de las diferentes sesiones necesarios para el entrenamiento del *plugin*.
  - ***p\_train***: Una vez que se han analizado todos los flujos presentes en el tráfico de entrada, se realiza el entrenamiento, ejecutándose el conjunto de algoritmos necesarios para ello.

## 4.2. Análisis estructural de TIE

Definidos los diferentes bloques de los que estará compuesto el clasificador, queda señalar dónde se insertan estos bloques en la estructura lógica de TIE. Este análisis es necesario para determinar las relaciones con las funciones y funcionalidades ya incluidas en TIE y su adecuación a los requerimientos del sistema a implementar.

Para ello, se analizará el código de TIE busca con el objetivo de tener el máximo conocimiento de los bloques que se van a diseñar, así como de aquellas otras partes que puedan afectar al *plugin* que se pretende implementar.

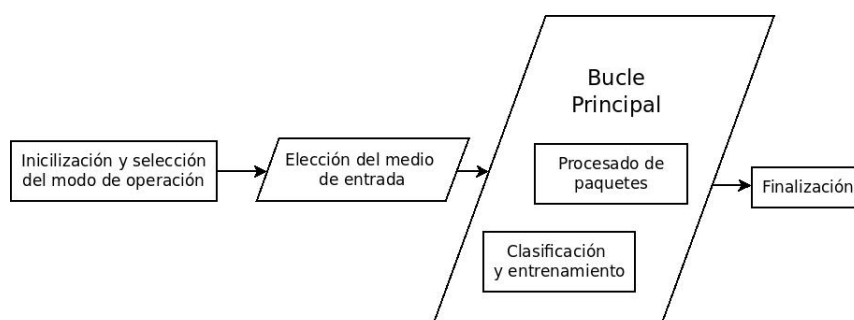


Figura 4.1: Principales bloques de TIE

De manera global, una ejecución de TIE pasa por cada una de las siguientes fases (Figura 4.1):

1. **Inicialización.** Se leen los argumentos indicados en la orden de ejecución de TIE, habilitándose, posteriormente, los clasificadores solicitados en dichos argumentos.
2. **Elección del medio de entrada.** Fija el medio de entrada, recopilando información relevante a cerca de ese elemento como el protocolo de la capa de enlace utilizado.
3. **Bucle principal.** Se encarga de ir leyendo cada uno de los paquetes del medio de entrada. Los dos siguientes bloques estarán contenidos en este, pues cada paquete leído por este bloque será analizado por los dos siguientes, dándose por finalizado este bloque cuando se han analizado todos los paquetes especificados.
4. **Procesado de Paquetes.** Se estudia el paquete, actualizándose la estructura de sesiones que permite a TIE almacenar el conjunto de flujos que está analizando.
5. **Clasificación y Entrenamiento.** Se clasifica, si es posible, la sesión a la que pertenece el paquete tomado por el bucle principal o se extraen los parámetros requeridos de dicho paquete para el entrenamiento posterior.
6. **Finalización.** Se imprimen los resultados de la clasificación realizada o se generan los modelos de entrenamiento en función del conjunto de muestras recopilado.

#### 4.2.1. Inicialización y selección del modo de operación

Siguiendo el flujo de ejecución del programa, los principales bloques que incorpora TIE en la fase de inicialización son (Figura 4.2):

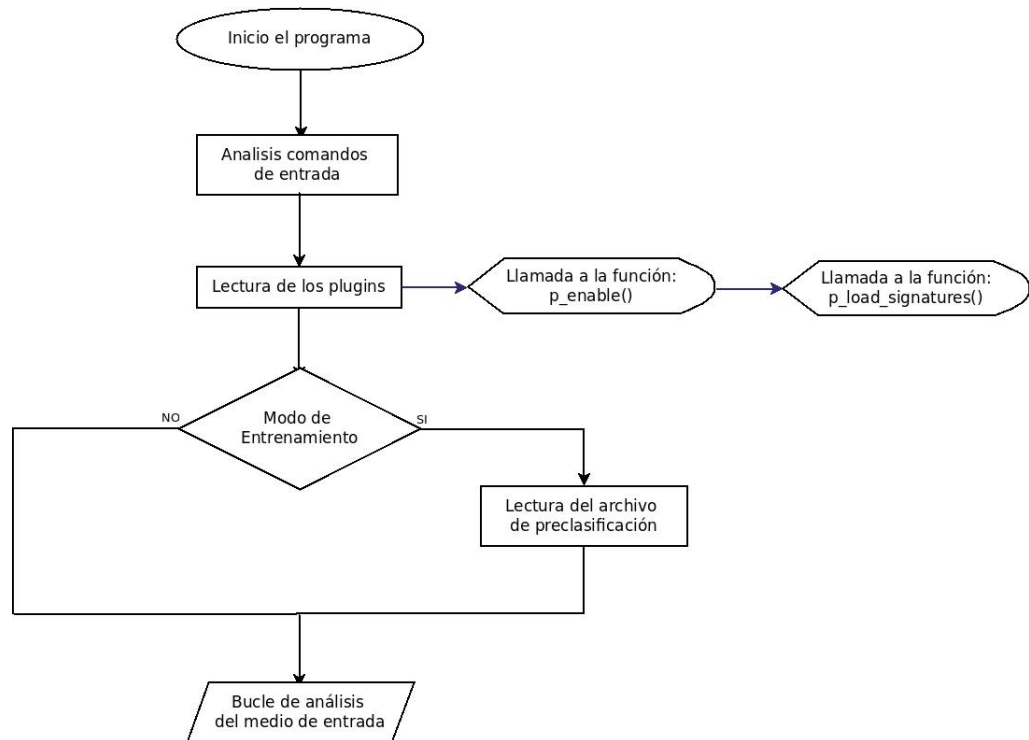


Figura 4.2: Bloques lógicos de TIE: Inicialización y selección del modo de operación

1. **Análisis de los argumentos de entrada.** Se interpretan cada uno de los argumentos y argumentos indicados a TIE. Algunos de los argumentos más significativos fueron expuestos en el Apartado 2.3.
2. **Lectura de los *plugins*.** Se cargan las referencias a los diferentes clasificadores que se hayan indicado en la línea de comandos. Cabe mencionar que durante la compilación de TIE se pueden anexionar diferentes *plugins* para que estén disponibles, mientras que será durante la fase de ejecución cuando se seleccionen los que se necesitan, no activándose los restantes para minimizar el coste computacional.

Es aquí donde se realiza la llamada a la función **p\_enable()**, comprobando que los parámetros necesarios para el *plugin* han sido solicitados. Seguidamente, se realiza la llamada a la función **p\_load\_signatures()** que servirá para cargar los modelos y/o parámetros de cada una de las aplicaciones caracterizadas en el *plugin*.

3. **Modo de entrenamiento.** Posteriormente, si el objetivo de la ejecución es realizar el entrenamiento de los modelos de un *plugin* se deberá cargar un archivo de preclasificación, también conocido como

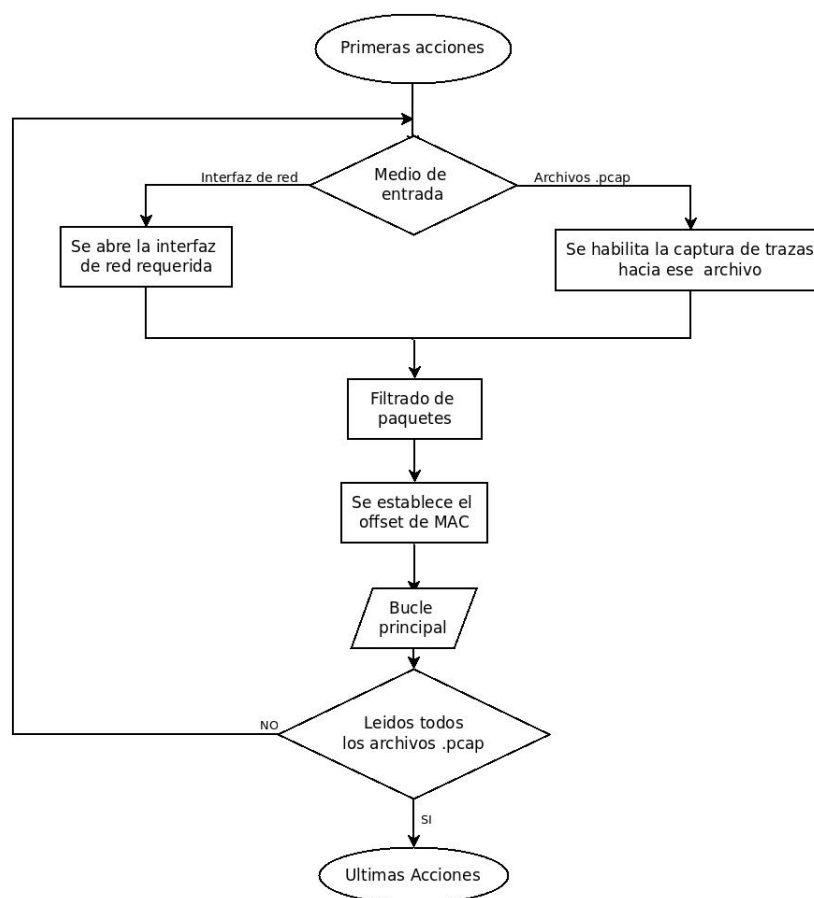


Figura 4.3: Bloques lógicos de TIE: Medios de entrada

*ground truth file.*

Generalmente, este archivo debe ser generado por otro *plugin* de TIE, el cual ha procesado previamente el tráfico con el que se va a entrenar el clasificador, asignando a cada sesión la aplicación de la que procede.

#### 4.2.2. Elección del medio de entrada

Una vez iniciada la plataforma, esta debe encargarse de caracterizar cada una de las distintas sesiones, para lo que habilitará la lectura de los paquetes con dichas sesiones. Las acciones a realizar son (Figura 4.3):

1. Se selecciona el medio de entrada, es decir, o se activa la captura de datos a partir de una interfaz de red para la entrada de las tramas o se habilita la lectura de un archivo en formato *libcap*.
2. Se establece un filtrado de los paquetes que se analizarán. Este filtrado se puede indicar desde la línea de comandos o desde un archivo externo,

Primitivas de filtrado BFP	
Comando	Significado
<b>dst host</b>	Filtrado para la IP del host de destino
<b>src host</b>	Filtrado para la IP del host de origen
<b>host</b>	Filtrado para IP tanto de destino como de origen
<b>dst port</b>	Filtrado para el puerto del host de destino
<b>src port</b>	Filtrado para el puerto del host de origen
<b>ip protocol</b>	Filtrado para paquetes IP

Tabla 4.1: Ejemplos de primitivas de filtrado BPF

usando para ello un conjunto de primitivas definidas en el lenguaje BPF *Berkeley Packet filter* [16], de amplio uso para este fin. Algunas de estas primitivas se muestran en la Tabla 4.1.

3. Se analiza el tipo de interfaz utilizado en la capa de enlace para definir el offset de la capa de red. Este valor es esencial, ya que servirá para indicar en pasos posteriores a partir de qué byte se tiene ya la parte perteneciente a la carga útil. En la versión de TIE actual se distinguen hasta tres tipos:
  - Ethernet
  - Raw
  - Linux\_SLL
4. Una vez establecido el modo de adquisición de datos, se ejecuta el bucle principal, encargado de leer los diferentes datos(paquetes/tramas) para su posterior análisis. Su operación se detalla en el apartado siguiente.
5. En caso de estar leyendo los paquetes desde un archivo *pcap* se deberá repetir el proceso con el siguiente archivo, hasta que estén todos leídos. En caso de acabar el bucle se procederá a terminar con la ejecución del programa.

### 4.2.3. Bucle principal

Este bucle engloba a las acciones que se le han de realizar a cada uno de los paquetes que se quiera analizar. Algunas de las acciones más importantes son (Figura 4.4):

1. Se toma el siguiente paquete a analizar, ya sea desde la interfaz de red o desde un archivo *pcap*.
2. Se determina si el paquete se puede procesar. Esto se realiza mediante un filtrado, evitando procesar aquellos paquetes que puedan ser pro-

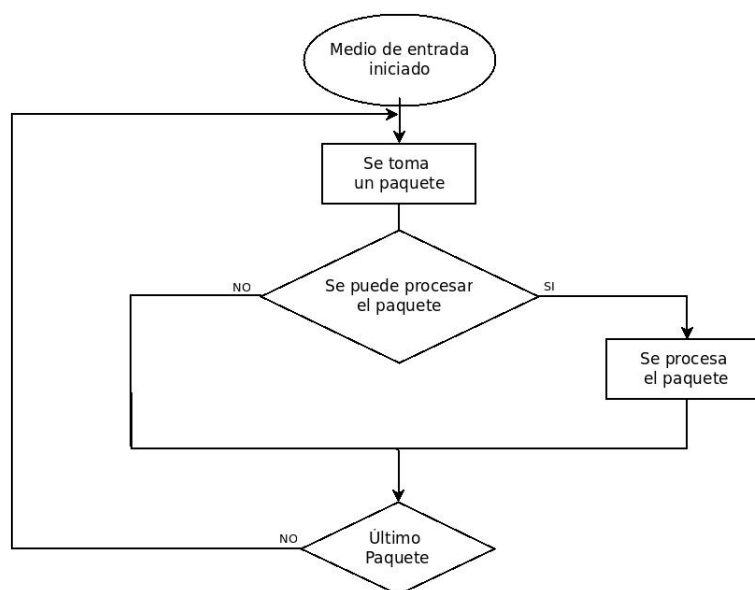


Figura 4.4: Bloques lógicos de TIE: Bucle principal

blemáticos, como pueden ser aquellos que estén fragmentados o aquellos que tengan cabeceras IPv4 opcionales, ya que la versión actual de TIE no puede procesarlas.

3. En caso de poder ser procesado, el paquete se analizará actualizando los valores de la sesión del flujo al que pertenezca el paquete en cuestión. Este conjunto de acciones se explicarán detalladamente en el siguiente capítulo.
4. Se reinicia el bucle en caso de no haberse leído todavía el último paquete del archivo *pcap*. Si se está leyendo tráfico desde una interfaz de red, se esperará hasta que llegue el próximo paquete o finalizará si se recibe una señal de interrupción.

#### 4.2.4. Procesado del paquete

Este bloque agrupa a los procesos encargados de evaluar el paquete y de actualizar los valores asociados a cada sesión. Estos son??:

1. Se determina el protocolo de la capa de transporte. Esta análisis es necesario a la hora de determinar donde comienza la carga útil de cada paquete.
2. Se valora si es necesario crear una nueva sesión para el flujo al que pertenece el paquete en cuestión. Una nueva sesión será necesaria si no

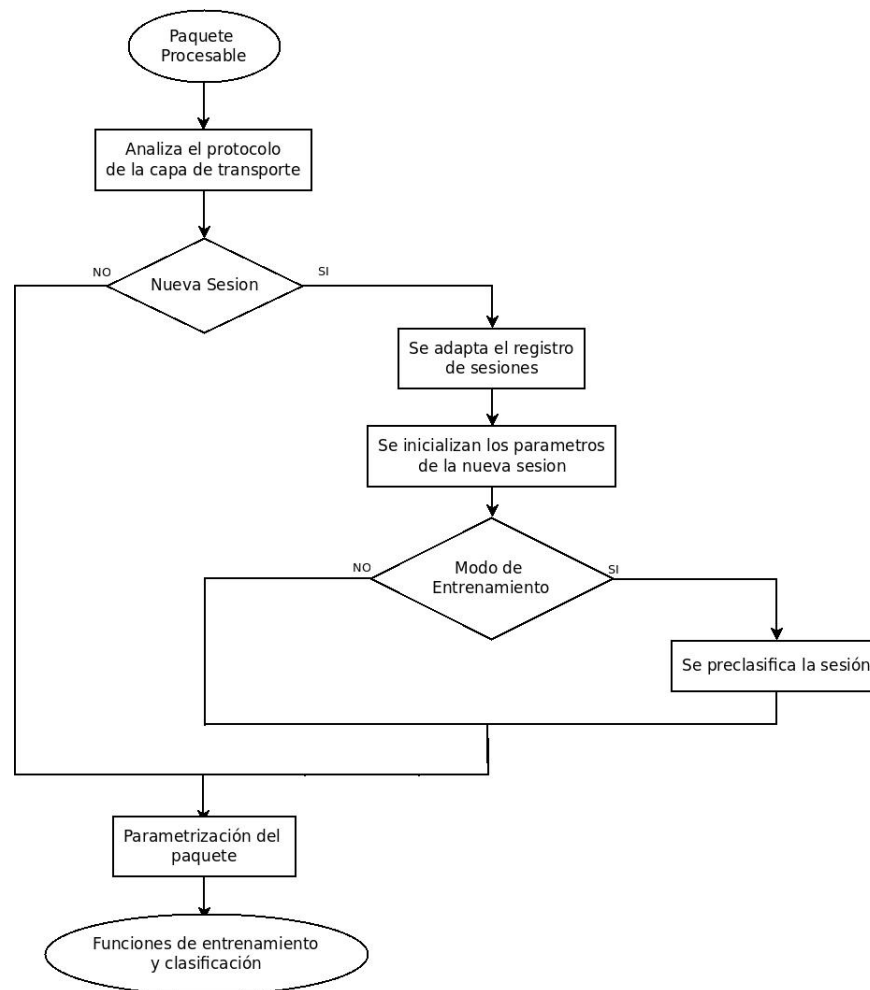


Figura 4.5: Bloques lógicos de TIE: Procesado del paquete

existe un registro asociado previamente, ha pasado un tiempo determinado o, en caso de ser TCP, si se han recibido previamente paquetes de desconexión.

3. Se actualiza el registro donde se almacenan las distintas sesiones para los diferentes flujos, así como las estadísticas establecidas por TIE.
4. Se inician los parámetros de la nueva sesión.
5. En caso de estar ejecutándose el entrenamiento de un *plugin*, se emparejará esta nueva sesión con su par cargado durante la fase inicial del programa desde un archivo de preclasificación.
6. Se analizan las características del paquete, guardándose aquellas que sean requeridas en los argumentos de la ejecución de TIE. Algunos de



los parámetros que TIE puede almacenar son:

- Tamaño de carga útil de los primeros  $N$  paquetes.
- Los primeros  $N$  bytes de una sesión.
- Valores temporales, como el instante de llegada de los paquetes.

Es aquí donde se ubica el extractor de parámetros de TIE, que será un bloque funcional que será necesario modificar, ya que no incluye todos los parámetros necesarios. En particular, no es posible extraer el tamaño de los mensajes mediante las funcionalidades ya disponibles.

#### 4.2.5. Clasificación y entrenamiento

Una vez que se han parametrizado las características requeridas del paquete, se continúa con una serie de acciones (Figura 4.6, relacionadas con la clasificación y el entrenamiento) :

- Si se está en modo de clasificación, se comprueba si la sesión a la que pertenece el paquete está dispuesta para ser clasificada con la función *p\_is\_session\_classifiable*. Si esto es así, se procede a clasificar la sesión mediante la función *p\_classify\_session()*
- Si se está en modo de entrenamiento, se registran los parámetros necesarios para el posterior entrenamiento, esto es, se aplica la función *p\_session\_sign()*.

#### 4.2.6. Finalización

Una vez que concluye el proceso de analizar todas las tramas que se hayan indicado desde la línea de comandos, TIE realiza una serie de acciones para terminar con la ejecución que se llevaba a cabo (Figura 4.7).

1. Se genera el archivo de salida que contendrá diferentes parámetros, como el tiempo que se ha tardado en llevar a cabo el análisis, así como los datos de la efectividad de los clasificadores empleados.

En esta función se hace una llamada a *p\_dump\_statistics()* donde simplemente se imprimirá el número de sesiones clasificadas, así como el total de aquellas que no se han podido clasificar.

2. Si el modo de clasificación ha sido el seleccionado, se imprimirá el archivo de clasificación que registra los resultados. El nombre de este archivo por defecto es *class.tie*, mediante los argumentos de línea de comando 2.3.

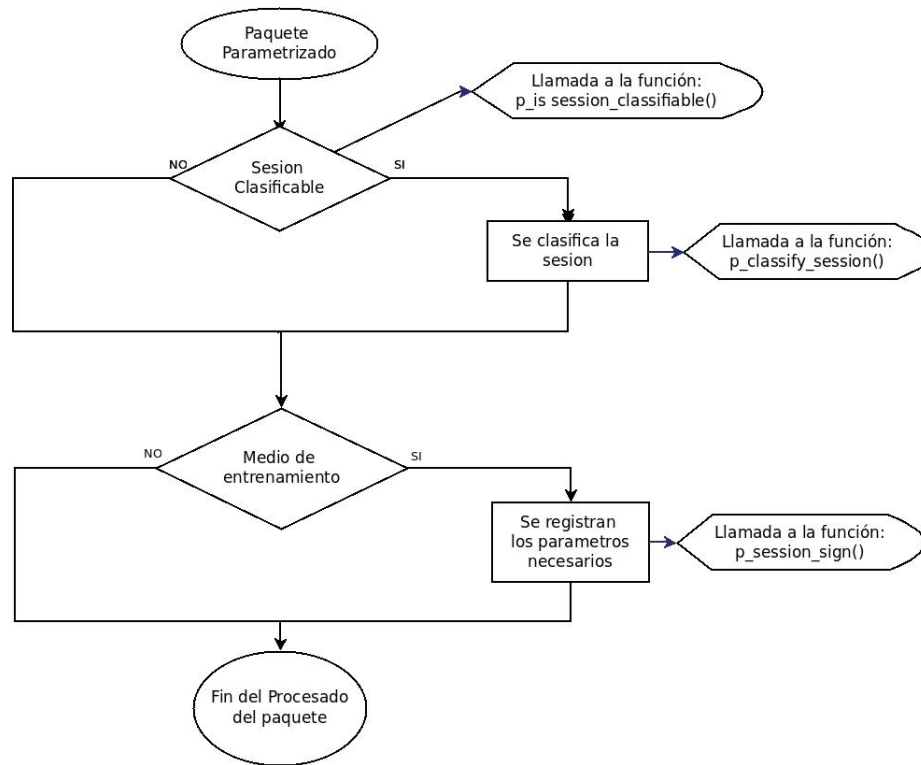


Figura 4.6: Bloques lógicos de TIE: Entrenamiento y clasificación

3. Si se indicó el entrenamiento del *plugin*, se realizará la llamada a la función ***p\_train()***, encargada de aplicar el algoritmo de entrenamiento correspondiente sobre el conjunto de muestras recogidas a lo largo del proceso.

Una vez analizado de forma global el funcionamiento de TIE se comenzará el diseño e implementación del clasificador. Como se indicó durante el Apartado 4.1, los bloques lógicos a desarrollar se dividen en las funciones de activación, las de clasificación y las de entrenamiento. Sin embargo, para obtener los parámetros necesarios para el funcionamiento del *plugin* a implementar, se necesita estudiar como se extraerán dichos parámetros, ya que TIE, en su versión actual, no cuenta con los mecanismos de extracción de esas características.

### 4.3. Parametrización

Lo primero a plantear es conseguir el conjunto de parámetros necesarios para llevar a cabo la clasificación de acuerdo al método propuesto. Tal y como se indicó en el capítulo 3, el método a implementar se fundamenta en estos elementos

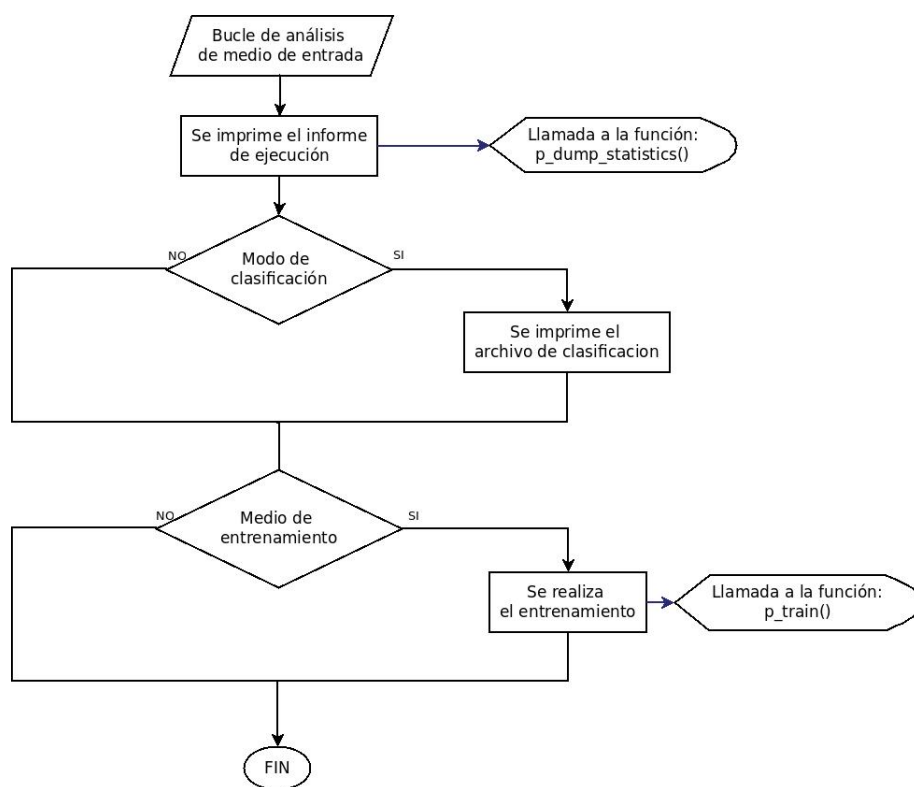


Figura 4.7: Bloques lógicos de TIE: Finalización

- El tamaño de los mensaje, que debe obtenerse acumulando el de la carga útil de los paquetes que lo constituyen.
- Criterios para la división de mensajes. Estos son (véase Apartado 3.2.1):
  - Para una sesión TCP:
    1. El flag de TCP PSH.
    2. Se considera que el tamaño de un la carga útil por debajo de un determinado valor es indicativo de que en ese paquete termina el mensaje actual.
    3. La dirección del paquete es contraria a la del paquete anterior.
  - Para una sesión UDP, todo paquete se considera un mensaje independiente a cualquier otro paquete.

De esto surgen dos problemas en relación a la actual implementación de TIE:

1. TIE cuenta con un único indicador por sesión para señalar si se ha detectado el flag PSH en alguno de los paquetes de dicha sesión. Por lo tanto, esta funcionalidad resulta insuficiente, ya que solo sería de utilidad para el primer mensaje de la sesión.
2. A priori, es imposible conocer de cuántos paquetes por sesión hace falta conocer su tamaño de carga útil, debido a que son los factores delimitadores del mensaje los que determinan cuándo acaba un mensaje, y cuándo se producirán estos factores es desconocido. a priori.

Consecuentemente, no se le puede indicar a TIE el número de tamaños de cargas útiles que se necesitarán, ya que o bien no son suficientes o hacen que se sobrecargue el sistema innecesariamente.

Por otro lado, se debe evitar modificar el código de TIE si no es totalmente necesario. Por ello, hace falta analizar detalladamente las posibles soluciones:

- **Evitando modificar el código de TIE:** No se tiene en cuenta el flag PSH y, por tanto, se deja al clasificador toda la responsabilidad de generar los mensajes, dados los tamaños de cargas útiles. Para esto se necesitaría analizar, de acuerdo a las funcionalidades de TIE, un número considerable de esos elementos.
- **Modificando levemente el código de TIE:** Se plantea registrar por cada paquete si está activado o no el flag PSH. De nuevo necesitaríamos un número considerable de cargas útiles.
- **Realizando una modificación sustancial del código de TIE:** Se delega en TIE la responsabilidad de determinar cual es el tamaño de los mensajes.

Tras valorar las diferentes opciones, se decide optar por la tercera opción, ya que cualquiera de las otras dos sobrecargaría en exceso al clasificador, provocando que la principal característica de este método que es la rapidez de identificación se viera totalmente comprometida.

Además, tal y como se explicará en las conclusiones de este trabajo, en el futuro TIE debería de dejar a los diferentes clasificadores la posibilidad de acceder al extractor de parámetros para así permitir que estos obtengan las características que necesiten.

Una vez establecido cómo se va a llevar a cabo la parametrización, se necesita valorar aquellos elementos de TIE que se deberán modificar:

- Se generarán nuevos tipos de argumentos de línea de comandos, relacionados con la parametrización para el *plugin*.
- Se definirá cómo se obtiene el tamaño de los mensajes en el bloque de extracción de características de TIE.

Comandos adicionales	
Comando	Parámetros que extra y almacena
<b>-u</b>	Tamaño mínimo de PDU de un paquete para no ser indicativo del final de un mensaje
<b>-U</b>	Número de mensajes necesarios por sesión

Tabla 4.2: Argumentos añadidos a TIE

- Se establecerán los nuevos parámetros necesarios para cada sesión para llevar a cabo la clasificación.

#### 4.3.1. Argumentos de entrada para parametrización

Se añadirán dos nuevos argumentos de línea de comandos para gestionar la parametrización a realizar (Tabla 4.2).

#### 4.3.2. Nuevos parámetros almacenados por sesión

TIE almacena un conjunto de variables para cada sesión en función de los argumentos indicados desde la línea de comandos. Para el sistema propuesto se necesitarán tres nuevas variables:

- Elemento **message\_count**: Un vector de tamaños de mensajes, que almacenará los tamaños de mensaje obtenidos para cada sesión, hasta el máximo prefijado. En las siguientes secciones también se le denominará como vector de muestras, ya que los tamaños de los mensajes son el espacio muestral con el que se trabaja.
- Elemento **message\_count\_lenght**: Un indicador de cuántos mensajes hay en el vector de tamaños. A su vez señala la posición del mensaje actual, al ser el último añadido al vector.
- Elemento **prev\_packet\_en**: Un indicador de si el mensaje actual<sup>1</sup>. almacenado en el vector de tamaños ha acabado. Esta variable es útil a la hora de estimar si un paquete pertenece al mensaje actual o si hace falta iniciar otro nuevo mensaje. La utilización de esta variable se muestra en el diagrama de flujo de la Figura 4.8.

En la Tabla 4.3 se resument estos nuevos parámetros, así como su funcionalidad de forma resumida.

<sup>1</sup>El mensaje actual es la última muestra del vector de tamaños de mensajes que se ha añadido. Viene señalada por el indicador del número de mensajes en el vector y se le deben añadir los futuros tamaños de cargas útiles hasta que se detecte la generación de un nuevo mensaje

Parámetros adicionales necesarios para el <i>plugin</i>		
Nombre	Tipo	Utilidad
<i>message_count</i>	u_int32	Vector de mensajes (vector de muestras)
<i>message_count_length</i>	*u_int8	Indice al último mensaje añadido
<i>prev_packet_end</i>	bool	Indica la finalización del ultimo mensaje añadido

Tabla 4.3: Parámetros añadidos a cada sesión

#### 4.3.2.1. Extracción del tamaño de mensaje

Esta fase es crucial para la implementación del clasificador, pues es la que se encargará de determinar los tamaños de los mensajes de cada sesión, con objeto de disponer de las muestras necesarias para llevar a cabo la clasificación.

El diagrama de flujo del procedimiento utilizado para la extracción de los tamaños de mensaje, de acuerdo con lo descrito en el Apartado 3.2, se muestra en la Figura 4.8. Los pasos más relevantes se detallan a continuación. En primer lugar, se evalúa si el paquete que ha llegado al extractor de parámetros es el primero de la sesión con datos. Las situaciones posibles son:

- No es el primer paquete con datos de la sesión. Se comprobará la dirección del paquete para así poder verificar si el mensaje actual tiene la misma dirección.

En caso de tener la misma dirección, se comprobará si el mensaje actual ya ha acabado mediante el parámetro dispuesto para este fin, (ver Apartado 4.3.2). De ser así, se iniciará<sup>2</sup> otro mensaje en el vector de muestras. En caso de no haber finalizado, se añadirá<sup>3</sup> el tamaño del contenido del paquete al del mensaje actual almacenado en el vector de muestras. En ambos casos se ha de respetar el criterio de signos.

En caso de tener dirección contraria se, inicia un nuevo mensaje en el vector de muestras, con un valor que será el del tamaño de la carga útil del paquete analizado (con signo negativo para *downstream* o positivo para *upstream*).

<sup>2</sup>Debe quedar claro que cuando se habla de iniciar otro mensaje se hace referencia a dar por evaluada una de las posiciones del vector de muestras, ya que el mensaje para esa posición ya está definido, al encontrarse un delimitador de mensaje, y comenzar la siguiente posición de la secuencia de mensajes

<sup>3</sup>Se suma el tamaño de la carga útil del paquete al tamaño que ya tenía el mensaje para esa posición del vector de muestras

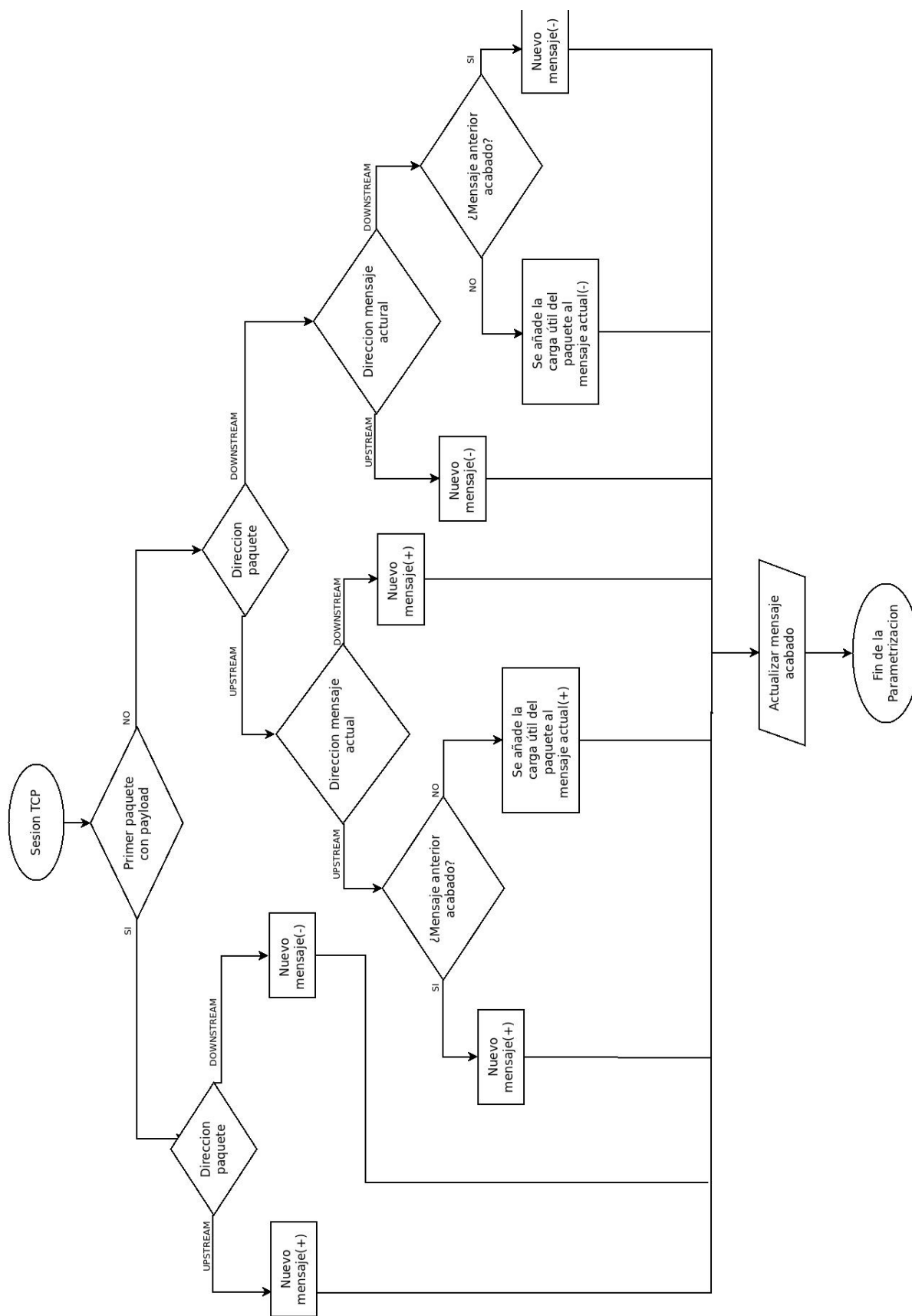


Figura 4.8: Esquema del procedimiento de extracción del mensaje

- Es el primer paquete con datos de la sesión: Se inicia el vector de muestras con el tamaño de la carga útil y con su respectivo signo, en función de la dirección del paquete.

En principio, ambas direcciones son posibles porque puede ser que fuera la entidad contraria a la que envía el primer paquete con contenido la que comenzara la comunicación con un paquete sin carga útil, quedando fijada la dirección por este primer intercambio.

Una vez extraídos los parámetros del paquete actual se deberá analizar si este paquete es el último de los que pertenecen al mensaje actual del vector de muestras. Será el paquete final del mensaje actual, activándose la variable *booleana prev\_packet\_end* (véase la Tabla 4.3), si el paquete cumple alguno de estos requisitos:

- La carga útil del paquete es inferior a la mínima fijada desde los argumentos de ejecución de TIE para no dar por evaluado el mensaje actual.
- Se detecta en dicho paquete el *flag* PSH.

Comentadas la serie de modificaciones realizadas para obtener los parámetros necesarios para el método que se va a implementar, se explicarán los diferentes bloques funcionales generados para la implementación del *plugin*.

## 4.4. Fase de activación

Esta fase engloba a las funciones necesarias para la puesta en marcha del clasificador, como son su habilitación así como la lectura de los modelos.

### 4.4.1. Clasificador habilitado

La primera función del clasificador a la cual TIE hará una llamada, como se explicó en el Apartado 4.1, es *p\_enable*, donde se deberá comprobar si se cumplen los requisitos para que el clasificador realice su función correctamente.

En este caso, se comprobará que se le han pasado por la línea de comandos los valores pertinentes del número de mensajes necesarios para la clasificación, así como el tamaño mínimo de PDU indicativo del final de un mensaje (véase Apartado 4.3.1).

Además, si el resultado es positivo, se cargará la estructura de valores globales para el *plugin*, valores que pueden ser modificadas desde el exterior a partir del archivo *MSBC\_conf.txt*. Para una descripción detallada de estos valores véase el Apéndice B.



#### 4.4.2. Lectura de modelos

Tras habilitar el clasificador, TIE realizará la lectura de las características de las aplicaciones que puedan ser identificadas por el método, esto es, de los modelos establecidos para cada aplicación. TIE cuenta con la posibilidad de deshabilitar esta lectura si solo se va a realizar un entrenamiento de los mismos.

Tal como se indicó en el Apartado 3.3 en el *plugin* que se pretende implementar debe haber un modelo general  $\lambda_n$  para cada una de las  $n$  aplicaciones que se pretende poder identificar.

Este modelo incorporará las distribuciones de probabilidad  $g(n, l)$ , que son diferentes para cada uno de los estados  $l$  (posición del mensaje en la secuencia), cuyo número total de estados,  $L$ , habrá sido determinado durante el entrenamiento.

A su vez, para cada estado se dispondrá dos grupos de distribuciones, uno para *upstream* y otro para *downstream*, por lo que, a modo de resumen, el modelado del fichero que contenga el conjunto de modelos, seguirá este esquema para los datos que incorpora:

- Número total de aplicaciones  $N$ 
  - Aplicación  $N^{\circ} Iden_0$ 
    - *Upstream*
      - Posición 0
        - Número de Gaussianas  $K$ 
          - $\mu_{C_{0,0,0}} \ w_{C_{0,0,0}} \ \sigma_{C_{0,0,0}}$
          - $\mu_{C_{0,0,1}} \ w_{C_{0,0,1}} \ \sigma_{C_{0,0,1}}$
          - .....
          - $\mu_{C_{0,0,k-1}} \ w_{C_{0,0,k-1}} \ \sigma_{C_{0,0,k-1}}$
      - Posición 1
        - Número de Gaussianas  $K$ 
          - $\mu_{C_{0,1,0}} \ w_{C_{0,1,0}} \ \sigma_{C_{0,1,0}}$
          - $\mu_{C_{0,1,1}} \ w_{C_{0,1,1}} \ \sigma_{C_{0,1,1}}$
          - .....
          - $\mu_{C_{0,1,k-1}} \ w_{C_{0,1,k-1}} \ \sigma_{C_{0,1,k-1}}$
      - Posición L-1
        - Número de Gaussianas  $K$ 
          - $\mu_{C_{0,L-1,0}} \ w_{C_{0,L-1,0}} \ \sigma_{C_{0,L-1,0}}$
          - $\mu_{C_{0,L-1,1}} \ w_{C_{0,L-1,1}} \ \sigma_{C_{0,L-1,1}}$
          - .....
          - $\mu_{C_{0,L-1,k-1}} \ w_{C_{0,L-1,k-1}} \ \sigma_{C_{0,L-1,k-1}}$

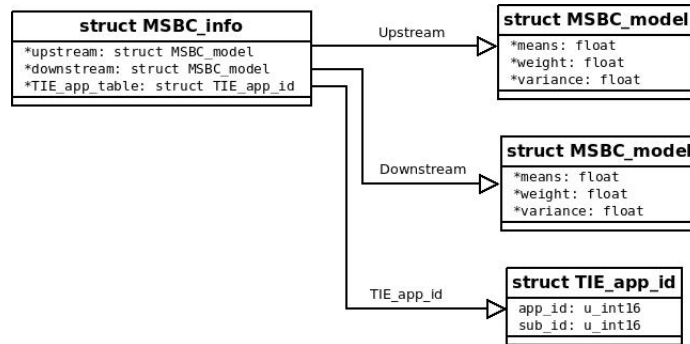


Figura 4.9: Estructura de almacenamiento del modelo para cada aplicación

- *Downstream*
  - Posicion 0
  - .....
  - Posicion L
- Aplicación  $N^{\circ}Iden_1$
- .....
- Aplicación  $N^{\circ}Iden_{N-1}$

El nombre de este fichero es *xxx\_MSBC\_signature\_file.txt*, donde *xxx* dependerá si el modelado a leer será el entrenado por el propio programa (*own*) o por un conjunto de modelos que han sido proporcionados desde otro programa (*foreign*).

El lector del fichero de clasificación ha sido diseñado con la utilidad de detectar todas aquellas aplicaciones para las que haya un estado sin clústeres para *upstream* o *downstream*, aunque esto puede ser natural si que puede ser un motivo de alarma. Durante la lectura de dicho fichero se almacenarán los datos en una serie de estructuras diseñadas para ese fin.

Estas estructuras serán del tipo *MSBC\_info* (véase la Figura 4.9), sirviendo cada una de ellas para almacenar el modelo de una aplicación. Por ello, hace falta tantas estructuras de este tipo como aplicaciones haya caracterizadas en el fichero *xxx\_MSBC\_signature\_file.txt*, construyéndose para ello un vector de estas estructuras, de tantas posiciones como aplicaciones haya definidas en el fichero de lectura.

Otro punto importante es que se deberá relacionar la nomenclatura utilizada por TIE con la nomenclatura utilizada por el programa que haya creado el fichero de clasificación.

Como se ha explicado durante la introducción a TIE en el Apartado 2.3, TIE identifica cada aplicación con un número, *AppId*, y dentro de cada

#N	App_ID	Sub_
0,	0,	0,
1,	1,	0,
2,	1,	1,
3,	1,	2,
4,	1,	3,
5,	1,	4,
6,	1,	5,
7,	1,	6,
8,	1,	7,
9,	1,	8,
10,	1,	9,
11,	1,	10,
12,	1,	11,
13,	1,	12,
14,	1,	13,
15,	2,	0,
16,	3,	0,
17,	4,	0,
18,	5,	0,
19,	6,	0,
20,	7,	0,

Figura 4.10: Extracto del fichero usado para asociar la identidad de cada aplicación con el conjunto de identificadores de TIE

aplicación distingue subtipos de aplicaciones con otro número más, *SubId*. En el fichero para los modelos especificado anteriormente solo se indicará con un número de que aplicación se van a incluir sus distribuciones, lo que hace necesario otro fichero que indique la relación entre este número identificador y su equivalente en *AppId* y *SubId* de TIE.

El fichero para llevar a cabo dicha asociación es llamado *xxx\_MSBC\_map\_file.txt* donde de nuevo *xxx* sirve para indicar la procedencia de los modelos. Este archivo ha sido creado manualmente en el caso de las distribuciones que han sido dados, *foreign*, y creado automáticamente por TIE durante el entrenamiento si estos son propios, *own*. En este fichero (Figura 4.10) se asociará en cada línea, el número de identificación, *#N*, usado en el fichero *xxx\_MSBC\_signature\_file.txt*, *nºIden<sub>n</sub>*, con sus equivalentes en los identificadores de TIE, *App-id* y *Sub-id*.

A la hora de llevar a cabo la relación entre la distintas identificaciones, debe quedar claro que en este fichero deben estar contenidas todas las aplicaciones que se puedan identificar con el *plugin*, aunque no se disponga todavía de modelos para esas aplicaciones.

Por ejemplo, para los modelos generados por el entrenamiento propio se han definido cada una de las distintas aplicaciones que actualmente TIE puede identificar, es decir, que existen número identificativos para esa aplicación en TIE. Para ello, se ha hecho uso del fichero de asociación de identidad de TIE *TIE.app.txt*, donde si una aplicación se dividía en *N* subaplicaciones en nuestro fichero, *own\_MSBC\_map\_file.txt*, habrá *N* aplicaciones para una sola 'aplicación' de TIE (*app-ID*). En este caso se han definido hasta un total de

410 aplicaciones, cada una con su número identificación,  $N^{\circ}Iden$ , diferente.

Por otra parte, para los modelos dados desde una plataforma externa venían definidos hasta 37 distintos tipos de aplicaciones, por lo que simplemente se ha mapeado este manualmente. Sin embargo, en el fichero de firmas, *foreign\_MSBC\_signature\_file.txt*, solo hay 10 tipos distintos de aplicación por lo que quedaban aplicaciones por caracterizar. El conjunto de estos modelos tuvo que ser previamente extraído, proceso que viene explicado en el Apéndice A.

## 4.5. Modo de clasificación

Este modo es el que se ejecuta cuando se quiere llevar a cabo la identificación del tráfico. Evidentemente, es necesario que previamente se hayan leído los modelos requeridos para este método de clasificación y se haya inicializado el *plugin* como se acaba de describir en el Apartado 4.4.2.

Esta fase consta de dos partes bien diferenciadas, pues TIE se encarga de pedir al *plugin* si la sesión esta lista para ser clasificada a través de la función **p\_session\_classifiable**, y si esto es así pasará a ser clasificada mediante la función **p\_classify\_session**.

### 4.5.1. Sesión Clasificable

Básicamente, se debe comprobar que la sesión está lista para ser clasificada. En principio esto será así si se han obtenido el número de mensajes indicados a TIE para poder realizarse la clasificación. Este número es un argumento que se le pasa a TIE con el comando -U, (véase tabla 4.2).

Sin embargo, para algunas aplicaciones, como por ejemplo DNS, son muy pocos los mensajes que se intercambian al principio de la comunicación antes de la descarga del contenido, por lo que es posible que no se llegue a conseguir ese número de mensajes necesario. Esto puede provocar que muchas aplicaciones no se clasifiquen debido a que no se llega a ese número de mensajes que habilita la clasificación.

Ante esta problemática, la solución por la que se ha optado ha sido realizar una clasificación cada vez que queda evaluado un nuevo mensaje para una sesión, reclasificandose de nuevo dicha sesión si queda definido otro nuevo mensaje.

Para llevar esto a cabo, cada vez que se active la variable *prev\_packet\_end* (ver tabla 4.3) presente en cada sesión será indicativo de que hay un nuevo mensaje y que la sesión puede clasificarse. Sin embargo, esta no será la clasificación final, puesto que se repetirá cada vez que quede definido un nuevo mensaje para esa sesión, terminando de realizarse este proceso cuando se llegue al número de mensajes especificado inicialmente como ideal para llevarse a cabo la clasificación.

Evidentemente, este modo de operación difiere del descrito en [3], ya que no todos los flujos se clasificarán considerando el mismo número de mensajes, y por tanto, la fiabilidad de la clasificación no será la misma para todos ellos. Se ha preferido esta solución para posibilitar la clasificación del mayor número posible de flujos y por considerar esta forma de operar más fácil de implementar en escenarios reales.

No obstante, la variable *prev\_packet\_end* puede no indicar el inicio de un nuevo mensaje en el caso de que sea debido a un cambio de dirección y no a un tamaño de carga útil por debajo del mínimo o a la presencia del flag PSH de TCP. Por esto, sería necesario incluir otra variable en cada sesión para poder definir esta otra posibilidad.

En resumen, la función **p\_session\_classifiable** devolverá *TRUE* cada vez que se active la variable *prev\_packet\_end*.

#### 4.5.2. Clasificación de sesión

Comprobado que la sesión está dispuesta para ser clasificada queda realizar sobre ella el algoritmo de clasificación. En el Apartado 3.3 se explicó el mecanismo que se lleva a cabo a la hora de identificar el tráfico, siendo determinante la posición en la que se reciben los mensajes.

A modo de resumen se mencionarán los principales elementos de este algoritmo. Básicamente, lo que se hace es comprobar la probabilidad,  $P(O_i|\lambda_n)$ , de que el modelo de una aplicación,  $\lambda_n$ , haya generado las observaciones,  $O_i$ , pertenecientes a un flujo,  $F_i$ .

Para ello, se recorrerán los modelos de estas aplicaciones,  $\lambda_n$ , calculando la probabilidad anterior para cada uno de ellos. Como se describió en el Apartado 3.3.1, la posición de los observables es determinante para evaluación de la probabilidad  $P(O_i|\lambda_n)$  final tal que:

$$P(O_i|\lambda_n) = \prod_{l=1}^L P(s_l|g(n, l)) \quad (4.1)$$

donde hay que calcular la probabilidad de que la distribución  $g(n, l)$  haya generado ese mensaje,  $s_l$ , en la posición  $l$  de la secuencia. Esa probabilidad será la suma de las probabilidades del mensaje  $s_i$  de acuerdo a la distribución gaussiana multiplico estimada durante el entrenamiento,

$$P(s_i|g(n, l)) = \sum_{i=1}^{K_{n,l}} w_{C_{i,n,l}} * P_{C_{i,n,l}}(s_l) \quad (4.2)$$

La probabilidad para cada una de las gaussianas,  $P_{C_{i,n,l}}(s_l)$ , se evalúa considerando la definición de distancia dada en el Apartado 3.2.2 de acuerdo a:

$$P_{C_{i,n,l}}(s_l) = \frac{1}{\sigma_{C_{i,n,l}}\sqrt{2\pi}} * e^{-0.5 \left[ \frac{D(s_l, \mu_{C_{i,n,l}})}{\sigma_{C_{i,n,l}}} \right]^2} \quad (4.3)$$

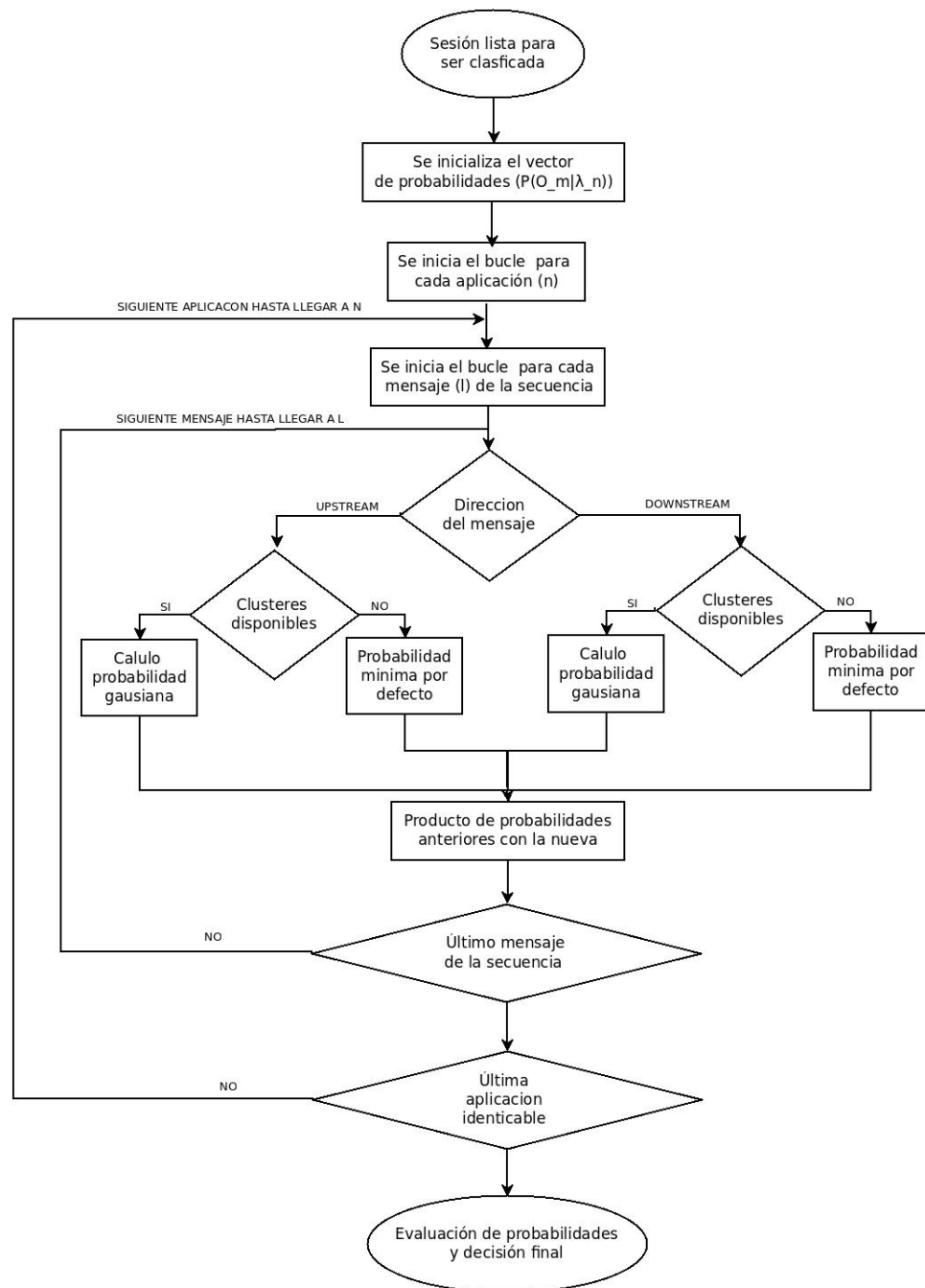


Figura 4.11: Procedimiento de clasificación

en la Figura 4.11 se muestra un esquema del núcleo del proceso, que se puede desglosar en los siguientes bloques:

1. Se inicia un vector que contendrá las probabilidades para todas las aplicaciones,  $P(O_i|\lambda_n)$ , con el valor 1, para que en principio todas las aplicaciones sean equiprobables. Este vector será en el que se almacene el productorio de la expresión 4.1.
2. Se crea un bucle que recorra los  $N$  modelos de todas las aplicaciones,  $\lambda_n$ . El índice de este bucle servirá de para indicar la posición del vector creado en el paso anterior.
3. Se crea otro bucle que recorrerá cada uno de los mensajes del vector de muestras,  $O_i$ .

4. Dado un mensaje,  $s_l$ , en la posición  $l$ , se comprueba su dirección y se evalúa si para esa posición y dirección hay distribuciones en el modelo.

En caso de no ser así, se le asigna una probabilidad,  $P(s_l|g(n, l))$ , por defecto para este mensaje. Aunque la probabilidad dada tendrá un valor mínimo, se pretende evitar, que por una posibilidad no definida en el modelo para esa aplicación, se anule la probabilidad global al multiplicar por 0. No obstante este factor puede modificarse desde el fichero de configuración, véase el Apéndice C.

En caso de existir distribuciones definidos se procederá a calcular la probabilidad de este mensaje aplicando la expresión 4.2.

Para el cálculo de esta probabilidad, además, se ha definido otra opción en la cual se calcula  $P(s_l|g(n, l))$  como el máximo valor de probabilidad entre todos las gaussianas (picos) y no como la suma de las probabilidades, esta posibilidad ha de indicarse desde el fichero de configuración externa (véase el Apéndice C). La expresión resultante sería la siguiente:

$$P(s_i|g(n, l)) = \operatorname{argmax}\{w_{C_{i,n,l}} * P_{C_{i,n,l}}(s_l) \quad 0 \leq i \leq K - 1\} \quad (4.4)$$

aquí deberá emplearse la expresión 4.3 para el cálculo de  $P_{C_{i,n,l}}$  para las diferentes gaussianas para ese modelo,  $g(n, l)$ .

5. Calculada la probabilidad  $P(s_l|g(n, l))$ , se debe actualizar el valor de probabilidad global correspondiente al modelo  $P(O_i|\lambda_n)$ , por lo que se realizará el producto de la probabilidad calculada con el valor ya almacenado. En caso de ser la primera iteración, el valor contenido previamente sería uno.
6. Este proceso se realizará para todas las aplicaciones que tengan definidas distribuciones y dentro de cada aplicación, para todos los mensajes del vector de observables hasta que concluyan ambos bucles. La sesión se etiquetará con aquella aplicación que haya obtenida una probabilidad  $P(O_i|\lambda_n)$  mayor.

Por último, se comprobará si el número de mensajes en el vector de muestras es el seleccionado, valor que se le indica a TIE por línea de comandos (Tabla: 4.2), si esto no es así se indicará que la sesión se deberá reclasificar cuando otro mensaje quede definido en el vector de observables. El objetivo de este procedimiento, que era asegurar algún tipo de clasificación para todas las aplicaciones, quedó explicado en el apartado anterior

## 4.6. Modo de entrenamiento

Este es el modo empleado para estimar los diferentes modelos necesarios por el *plugin* para llevar a cabo la identificación del tráfico. Este modo de entrenamiento ha sido diseñado con la idea de que todas las aplicaciones que actualmente están definidas en TIE puedan ser caracterizadas y, por lo tanto, posteriormente identificadas.

Sin embargo, para poder entrenar una aplicación hace falta tener durante el entrenamiento un conjunto de flujos pertenecientes y preetiquetados como generados por esa aplicación en número suficiente, con objeto de poder parametrizarlos y tener el conjunto de muestras necesario para el correcto entrenamiento de esa aplicación.

Para ello, TIE dispone de la posibilidad de generar el archivo con los flujos etiquetados, el denominado *ground\_truth\_file*, a partir de los otros *plugins* disponibles en la plataforma. De los clasificadores indicados durante el Apartado 2.3, openDPI aún se encuentra en fase beta de desarrollo, por lo que es un *plugin* muy inestable. Lo ideal, sería usar este método, mucho más fiable que los otros expuestos, para generar el archivo con las sesiones preclasificadas. Sin embargo, ante este problema se deberán buscar formas alternativas de conseguir este archivo con las sesiones etiquetadas. Durante el capítulo se expondrán los procedimientos empleados para solventar el problema anterior.

Como se detalla durante la exposición del método de entrenamiento en el Apartado 3.4, se realizará un mecanismo de agrupamiento sobre cada conjunto de muestras, cada uno correspondiente a una aplicación diferente.

La forma de obtener cada conjunto de muestras se realiza de forma similar al método de clasificación, es decir, mediante la extracción de parámetros siendo en este caso el tamaño de los  $L$  primeros mensajes de una sesión.

Una vez extraídos estos parámetros, deben ser almacenados en ficheros (o en la propia memoria de ejecución del programa) para poder realizarse posteriormente sobre ellos el algoritmo de *clustering*, K-medias en nuestra caso. La función encargada de almacenar los parámetros es **p\_session\_sign**. Esta función podemos dividirla en dos partes:

- Inicio del entrenamiento.



- Almacenamiento de parámetros.

Estas dos partes se describirán en dos capítulos por separado aunque estén contenidas en la misma función lógica de TIE.

#### 4.6.1. Inicio del proceso

El entrenamiento comenzará cuando llegue el primer paquete con contenido que pondrá en marcha el proceso en el que se cargarán una serie de estructuras relacionadas con el entrenamiento.

Tal y como se mencionó antes, se planteaban dos formas de almacenar estos parámetros:

- En la propia memoria de ejecución del programa. El problema asociado a esta forma de operar es que si el programa termina sin finalizar toda la parametrización realizada se perdería o, incluso aún peor, que se almacenen demasiados parámetros y que se colapse la memoria RAM del sistema.
- Mediante ficheros externos al programa. Este procedimiento es mas lento que el otro aunque en principio no es un factor determinante, ya que el entrenamiento no solo debe de ejecutarse una vez.

Finalmente ,como el tiempo empleado no es un factor limitante para el entrenamiento, se optó por la opción de los ficheros. Cada uno de estos ficheros deberá almacenar las muestras de tamaño de mensaje para una aplicación,  $n$ . Este fichero contendrá tanto las muestras de mensaje para dirección *downstream* como para dirección *upstream*.

Para la construcción y direccionamiento de estos ficheros se han creado una serie de estructuras las cuales se instancian durante esta fase inicial del entrenamiento. El esquema de esta estructura se detalla en la Figura 4.12.

El esquema de archivos se puede resumir en lo siguiente:

1. Existe un vector de estructuras tipo *tie\_app\_id\_files*. Cada una de estas estructuras se corresponde con un tipo de aplicación de TIE, como en TIE hay varias subaplicaciones definidas por aplicación, cada estructura *tie\_app\_id\_files* a su vez contendrá un vector de estructuras tipo *tie\_app\_sub\_id\_files*, una por cada subaplicación. El elemento *num\_app\_sub*, contenido en la estructura *tie\_app\_id\_files*, indica cuántas subaplicaciones hay en esa aplicación, esto es, el tamaño del vector de estructuras *tie\_app\_sub\_id\_files* contenido.
2. Cada estructura tipo *tie\_app\_sub\_id\_files* contendrá estos elementos:
  - Puntero *sub\_app\_file*, sirve como direccionamiento al fichero para la subaplicación en cuestión.

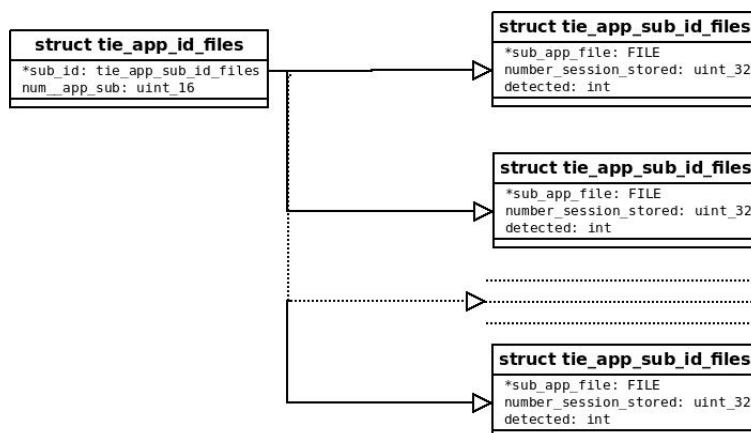


Figura 4.12: Estructura de almacenamiento de modelos para cada aplicación

- Elemento *number\_session\_stored*, Útil a la hora de reservar memoria para almacenar las diferentes muestras.
- Elemento *detected*, Usado para saber si hay alguna muestra perteneciente a esa subaplicación. Evita problemas al intentar acceder a un archivo sin ninguna muestra.

Para definir estas estructuras hace falta saber cuántas aplicaciones hay definidas en TIE. Para ello, se hace uso del archivo *TIE\_apps.txt*, que contiene el identificador de cada aplicación, el identificador de supaplicación, así como una breve descripción de la aplicación, (ver Figura 4.13).

En la Figura se observa la estructura de este fichero, así como que para una misma aplicación puede haber diferentes subaplicaciones como es el caso de HTTP. Este fichero será recorrido por un bucle con el objetivo de definir las distintas aplicaciones, ya que para cada aplicación hay distinto número de subaplicaciones.

Además de la principal funcionalidad expuesta anteriormente, durante esta fase de entrenamiento también se permite mostrar el archivo *own\_MSBC\_map\_file*, necesario para asociar las aplicaciones entrenadas a sus números de identificación en TIE, *app\_id* y *sub\_id*. Esta posibilidad puede ser deshabilitada desde el archivo de configuración si ya se cuenta con este fichero.

De esta forma a cada aplicación se le asigna un número identificativo en función de su posición en el archivo *TIE\_app.txt*. A modo de ejemplo, se incluye una captura de este otro fichero en la Figura 4.14.

La primera aplicación, la número 0, se corresponde con el grupo de desconocidas de TIE. Las 14 siguientes corresponden a HTTP, siendo la subaplicación 0 la HTTP genérica (véase la Figura 4.13).

Como ya se ha indicado, este archivo servirá para asociar la identificación de TIE, (*App-ID*, *Sub-id*), con su número asignado, *N*, que será la única

```

~
#AppID  SubID  GroupID Label SubLabel
0,      0,      0,      "UNKNOWN", "UNKNOWN",
#
1,      0,      1,      "HTTP",    "HTTP",
1,      1,      1,      "HTTP",    "DAP",
1,      2,      1,      "HTTP",    "FRESHDOWNLOAD",
1,      3,      1,      "HTTP",    "AUDIO",
1,      4,      1,      "HTTP",    "CACHEHIT",
1,      5,      1,      "HTTP",    "CACHEMISS",
1,      6,      1,      "HTTP",    "VIDEO",
1,      7,      1,      "HTTP",    "QUICKTIME",
#Added for OpenDPI compatibility
1,      8,      1,      "HTTP",    "OFF",
1,      9,      1,      "HTTP",    "AVI",
1,      10,     1,      "HTTP",    "MOVE",
1,      11,     1,      "HTTP",    "OGG",
1,      12,     1,      "HTTP",    "MPEG",
1,      13,     1,      "HTTP",    "HTTP_FLASH",
#
2,      0,      1,      "SQUID",   "SQUID",
3,      0,      1,      "SQUID_ICP", "SQUID_ICP",
4,      0,      1,      "HTTPS",   "HTTPS",
5,      0,      9,      "DNS",     "DNS",
5,      0,      9,      "MULTICAST_DNS", "MULTICAST_DNS",
7,      0,      9,      "OSUNMS",  "OSUNMS",
8,      0,      9,      "SLP",     "SLP",
9,      0,      9,      "BOOTP",   "BOOTP",
#
10,     0,      3,      "FTP",     "FTP",
10,     1,      3,      "FTP",     "FTP_DATA",
10,     2,      3,      "FTP",     "FTP_CONTROL",

```

Figura 4.13: Fragmento del fichero *TIE\_app.txt*

identificación que referencia a cada aplicación en el archivo que contenga el modelado de las diferentes aplicaciones, *own\_MSBC\_signature\_file.txt*.

#### 4.6.2. Almacenamiento de parámetros

Una vez inicializados los diferentes ficheros para el entrenamiento, se empezarán a recopilar los tamaños de mensaje de cada una de las aplicaciones que se pretende modelar. De esto se encarga la función **p\_session\_sign**.

Es aquí donde se comprueba la utilidad de la disposición de las estructuras planteadas en la Figura 4.12. Como se ha explicado anteriormente, cada aplicación preclasificada tendrá ya asignado su número de aplicación en TIE y su número de subaplicación, permitiéndonos navegar y ubicar cada aplicación en su fichero correspondiente utilizando estos valores como índices en los vectores de estructuras expuestos en dicha Figura.

Esta función imprimirá el tamaño de los primeros  $L$  mensajes para cada sesión en su fichero de almacenamiento correspondiente, *sub\_app\_file*, de la estructura *tie\_app\_sub\_id\_files*.

También se encargará de actualizar el valor de las variables *detected* y *number\_session\_stored* de esa estructura. Este proceso generará un conjunto de fichero como el de la Figura 4.15, cada uno de ellos referidos a las muestras

#N	App_ID	Sub_
0,	0,	0,
1,	1,	0,
2,	1,	1,
3,	1,	2,
4,	1,	3,
5,	1,	4,
6,	1,	5,
7,	1,	6,
8,	1,	7,
9,	1,	8,
10,	1,	9,
11,	1,	10,
12,	1,	11,
13,	1,	12,
14,	1,	13,
15,	2,	0,
16,	3,	0,
17,	4,	0,
18,	5,	0,
19,	6,	0,
20,	7,	0,

Figura 4.14: Archivo *own\_MSBC\_map\_file.txt*

recogidas para cada subaplicación de TIE.

En el archivo de la Figura 4.15 se tiene una configuración de 8 mensajes, donde hay tanto tamaños para dirección *upstream* como para dirección *downstream*, ya que será posteriormente cuando se separen las muestras positivas, dirección ascendente, de las negativas, dirección descendente.

#### 4.6.3. Ejecución del algoritmo de clustering

Esta funcionalidad corresponde a la parte principal del entrenamiento, ya que es donde se obtiene el modelado en sí de las diferentes distribuciones para cada una de las distintas aplicaciones. La función lógica de TIE encargada de hacer esto es **p\_train**, la cual es llamada cuando se han leído y evaluado todos los flujos del archivo o archivos *pcap* usados como medios de entrada de paquetes véase (la Figura 4.7).

Durante esta fase se recorrerán las diferentes muestras recopiladas, que

```
#Temporal file used to store the sizes of the messages
#Pos: 0 #Pos: 1 #Pos: 2 #Pos: 3 #Pos: 4 #Pos: 5 #Pos: 6 #Pos: 7
92,    -895,   92,    -895,   92,    -895,   92,    -895,
92,    -895,   92,    -895,   92,    -895,   92,    -895,
92,    92,    92,    -895,   92,    -895,   92,    -895,
92,    -895,   92,    -895,   92,    -895,   92,    -895,
92,    -895,   92,    -895,   92,    -895,   92,    -895,
```

Figura 4.15: Extracto de un fichero de recogida de muestras para el entrenamiento

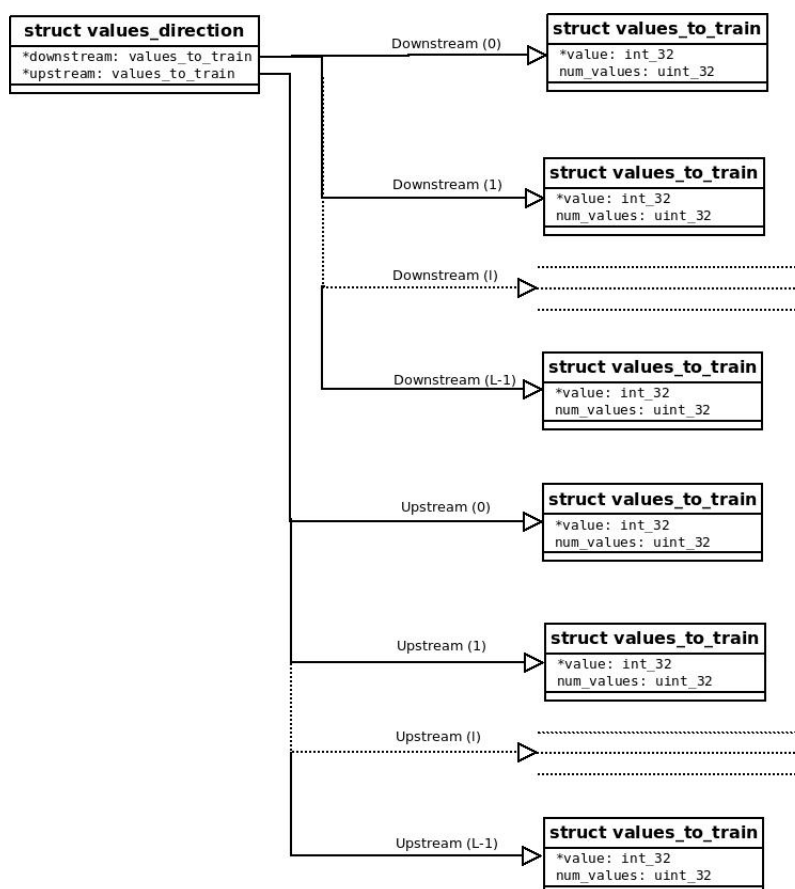


Figura 4.16: Estructura usada para almacenar las muestras de una aplicación

están almacenadas en los archivos como el de la Figura 4.15. Para aplicar el algoritmo de agrupamiento se deben ir seleccionando sucesivamente las muestras para una posición en la secuencia de mensaje,  $l$ , para una aplicación,  $n$ , y para ambas direcciones, aplicándose el algoritmo en cada una de estas combinaciones.

Por ello, se necesitan una serie de bucles así como de estructuras que permitan llevar a cabo esta selección. Las estructuras planteadas servirán para almacenar y clasificar las muestras de cada uno de los archivos antes mencionados, es decir, las muestras para una aplicación. Las estructuras planteadas siguen el formato de la Figura 4.16, que se puede resumir en lo siguiente:

- Se crea una estructura, *values\_direction*, que almacenará a otros dos vectores de estructura tipo *values\_to\_train*; uno para *upstream* y otro para *downstream*.
- A su vez, cada vector de estructuras tipo *values\_to\_train* servirá para

almacenar las muestras, para una única dirección *upstream* o *downstream*, de tamaño de mensaje para la aplicación, dada por el fichero que se está procesando, y para la posición en la secuencia, *l*, dada por su lugar en ese vector de estructuras.

- Por último, una estructura tipo *values\_to\_train* define estos elementos:
  - Un vector de enteros, llamado *values*, que almacenará todas las muestras para esa dirección, esa posición de secuencia y esa aplicación.
  - Un entero llamado *num\_values* que indique cuantas muestras hay en el vector *values*.

Examinada la forma en la que se van a almacenar las muestras de tamaño, es necesario exponer ahora cómo se recorrerán los diferentes archivos que contienen las muestras. Se crearán dos bucles enlazados:

- Uno recorrerá el vector de estructuras de aplicaciones<sup>4</sup> tipo *tie\_app\_id\_files*, seleccionando cada una de las distintas aplicaciones en cada iteración.
- El siguiente bucle tomará cada una de las distintas subaplicaciones definidas en el vector de estructuras tipo *tie\_app\_sub\_id\_files*, contenidas en la estructura anterior.

El proceso aquí expuesto queda representado en el diagrama de flujo de la Figura 4.17.

En este proceso de lectura de ficheros de cada subaplicación se usará el elemento *detected*, presente en toda estructura *tie\_app\_sub\_id\_files*, con objeto de evitar acceder a ficheros que estén vacíos, consiguiéndose así aligerar el entrenamiento.

Separados los espacios muestrales y cargada la estructura que servirá como referente para acceder a las muestras, se procederá con el algoritmo de agrupamiento. Pero antes, dado que con la estructura seleccionada se tiene tanto elementos para *upstream* como para *downstream*, así como las diferentes posiciones de mensaje para la secuencia, habrá que ir tomando cada espacio muestral de forma individual con nuevos bucles, esquematizados en la Figura 4.18.

---

<sup>4</sup>En este caso se trata de las aplicaciones definidas por TIE, es decir, sin tener en cuenta las subaplicaciones, ya que es esto lo que se tomó como referencia a la hora de definir estas estructuras.

Sin embargo, cuando se habla de las *N* aplicaciones identificables por el clasificador se hace referencia a todas las aplicaciones, es decir, la suma de todas las subaplicaciones en TIE.

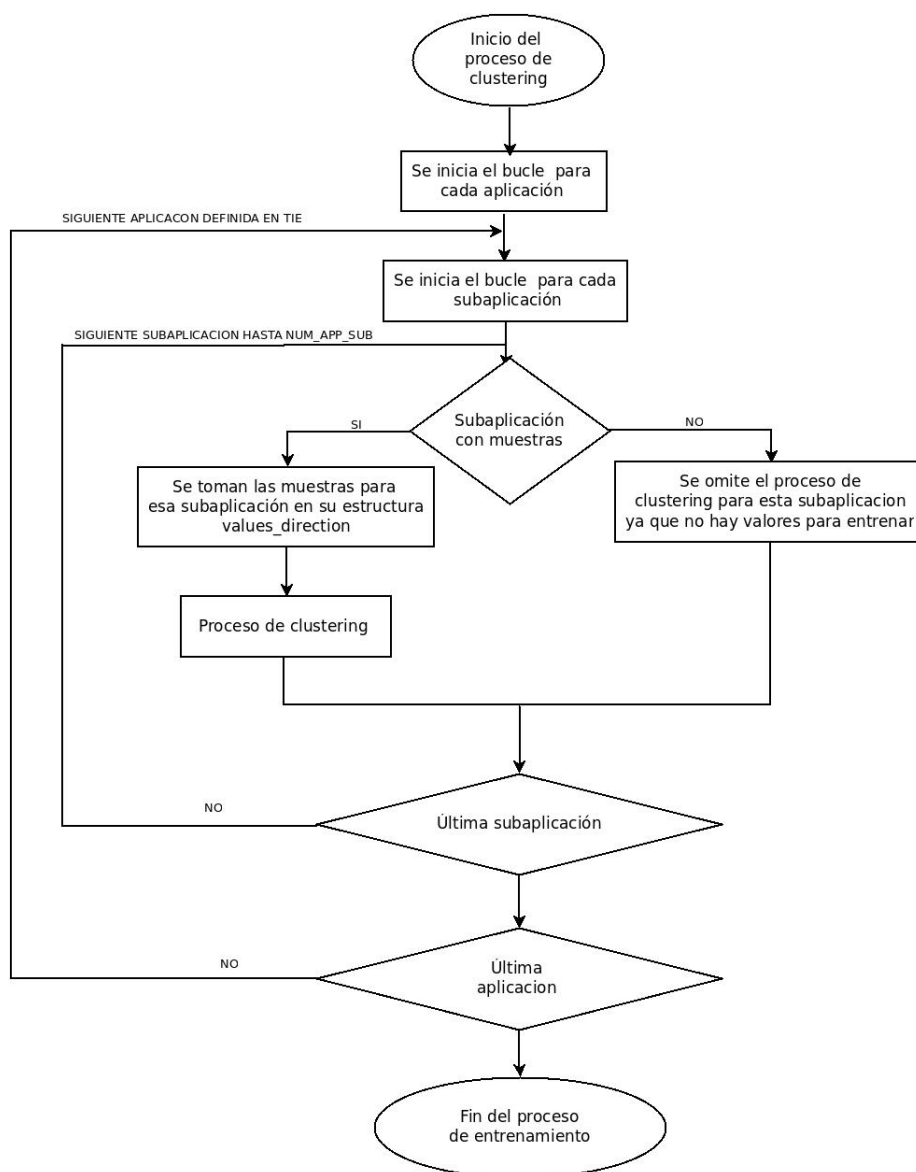


Figura 4.17: Arquitectura previa al algoritmo de agrupamiento

Uno de los bucles tendrá dos iteraciones, una para cada dirección<sup>5</sup>, mientras que el otro bucle será de  $L$  iteraciones, que son el número de posiciones de la secuencia de tamaños de mensaje, tomando en cada ronda las muestras que están almacenadas en la estructura tipo *values\_to\_train*, para una posición y en una dirección.

<sup>5</sup>No se trata de un bucle real, ya que lo que se hace es cambiar la dirección de un puntero a las estructuras *values\_to\_train*, una vez a *upstream* y luego a *downstream*, presentes en la estructura que se recorre, *values\_direction*

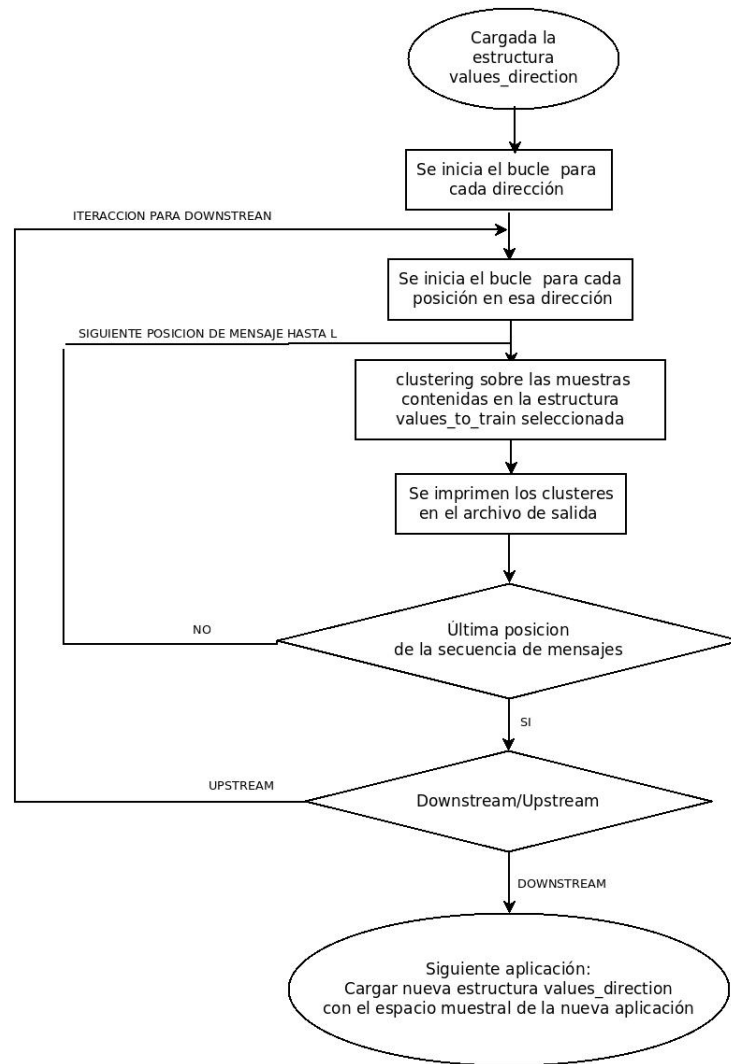


Figura 4.18: Estructura de selección de cada espacio muestral para una sola aplicación

Finalmente ya se tiene un espacio muestral, el vector de valores *num\_values* presente en la estructura *valores\_to\_train*, que era el objetivo que se perseguía con los procedimientos anteriores, consecuentemente ya se puede llevar a cabo el algoritmo de agrupamiento, para cada una de las aplicaciones y posiciones de mensaje.

#### 4.6.3.1. Algoritmo K-medias

Quedan por exponer los bloques funcionales en los que se ha dividido



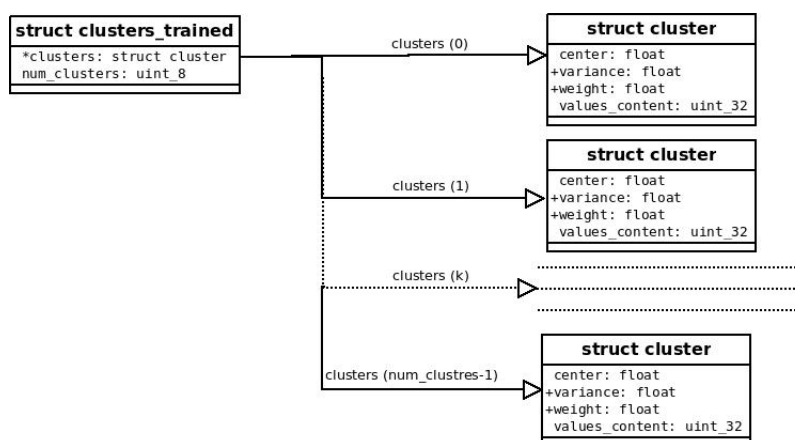


Figura 4.19: *struct clusters\_trained*. Estructura temporal usada durante el algoritmo de *clustering*, empleada en almacenar las diferentes distribuciones establecidas durante dicho proceso

el algoritmo de agrupamiento usado, que en este caso es K-medias. Este procedimiento ya fue descrito en el Apartado 3.4.1, aunque en este se pretende profundizar en la implementación realizada, así como en algunas de las variables que influyen en la ejecución del algoritmo.

#### 4.6.3.1.1. Variables y estructuras implicadas

En estos bloques se hará uso de una serie de estructuras útiles a la hora de organizar los diferentes modelos, así como para almacenar la asignación de las muestras a cada una de las gaussianas de los modelos durante las diferentes fases del algoritmo. La principal estructura es *cluster\_trained* (Figura 4.19) que servirá para referenciar a los distintos *clusters* generados por el algoritmo.

Los elementos que forman parte de esta estructura son:

- Un elemento denominado *num\_clusters* que registrará el número de *clusters* que se tienen almacenados. Durante el proceso de estimación se irá aumentando el número de *clusters* presentes, por lo que es importante saber hasta qué posición del vector hay *clusters* definidos.
- Un vector, denominado *clusters*, que contiene estructuras tipo *cluster*. Cada una de estas estructuras almacenará los parámetros y variables de los diferentes *clusters* que se obtenga durante el proceso. Para ello se definen los siguiente elementos en esta estructura:
  - Elemento *center*, almacenará el *centroide* del *clúster*.
  - Elemento *variance*, almacenará la *varianza* del *clúster*.

- Elemento *values\_content*, almacenará el número de muestras que pertenecen a ese *clúster*. Útil a la hora de estimar el peso relativo de cada *clúster*.
- Elemento *weight*, almacenará el peso de cada *clúster*, es decir, la importancia relativa de este *clúster* respecto a los demás del modelo.

Expuesta la estructura utilizada para manejar los *clusters*, también es necesario enunciar las diferentes variables que van a ser utilizadas para completar el algoritmo con exitoso:

1. Elemento *array\_belong*. Este parámetro se encarga de almacenar la pertenencia de cada muestra a un *clúster* o a otro.

Se trata de un vector de enteros que tendrá tantas posiciones como muestras haya en la ejecución que se este tratando, referenciándose cada posición a una de las muestras, es decir, la primera posición hará referencia a la primera muestra del vector *num\_values*<sup>6</sup>, indicando con un entero la posición que ocupa el *clúster* al que pertenece en el vector de estructuras *clusters*.

2. Elemento *empty\_cluster\_pos*. Como se verá más adelante, este entero indicará la posición de alguno de los *clusters* que se haya podido quedar vacío durante el proceso.

El objetivo de esta variable es evitar ir aumentando el número de *clusters* si otros se están quedando vacíos, pues se usará este valor como referencia del lugar que deberá ocupar el próximo *clúster* en el vector de estructuras tipo *cluster*, *clusters*.

3. Elemento *most\_spread\_cluster\_pos*. Una de las opciones consideradas para la bisección de *clusters* permitía únicamente dividir un *clúster* en cada iteración.

En este caso, es necesario saber qué *clúster* dividir por lo que se necesitará esta variable para referenciar dicha posición.

4. Elemento *n\_clusters\_active*. Esta variable almacena el número de *clusters* activos en cada instante. Como se ha mencionado anteriormente, durante la ejecución del algoritmo se aumentará progresivamente el número de *clusters*, por lo que hace falta registrar cuántos se tienen en cada repetición.

5. Elemento *estimated\_error*. Almacena el valor del error cuadrático actual,  $\xi$ . La forma de calcular este error se detallará más adelante.

---

<sup>6</sup>Figura 4.16, el vector *num\_values* está contenido en la estructura *values\_to\_train*.

6. Elemento *last\_estimated\_error*. Esta variable almacenará el error y servirá para comprobar si durante una repetición del algoritmo básico de K-medias, esto es, fijado el número de *clusters*, se ha reducido el error  $\xi$  al compararlo con *estimated\_error*. En la siguiente sección se explicará cómo cada vez que se aumenta el número de *clusters* se realizará un proceso K-medias básico<sup>7</sup>.
7. Elemento *up\_last\_estimated\_error*. Similar al anterior, pero esta referida al bucle de bisección<sup>8</sup> de los *clusters*. Almacena el error previo a la iteración y comprueba si se ha reducido este al compararlo con *estimated\_error*, que es el error actual.

Aparte de estas variables, se usan otros parámetros que regulan el comportamiento del algoritmo quedando definidos por defecto para el programa, aunque pueden ser modificadas desde el archivo de configuración *MSBC\_conf.txt*. Estas variables están definidas en la estructura tipo *MSBC\_variables*, siendo denominada *MSBCv* en la implementación, por lo que todas las variables tendrán el nombre *MSBCv.xxx* donde *xxx* será el nombre de la variable.

En el Apéndice B se indican cuales son estos valores por defecto con una explicación completa de para qué se emplean y cómo se pueden modificar desde el archivo de configuración externo al programa. Para la ejecución del algoritmo las dos más relevantes son:

1. Elemento *MSBCv.clusters\_required*. Indica el número máximo de *clusters* permitido para el modelado.
2. Elemento *MSBCv.minimum\_error\_recoverable*. Indica la mínima variación de error en una repetición que permitiría volver a ejecutar el bucle. Con objeto de evitar bucles infinitos, se implementa esta variable para evitar repetir un bucle si no se ha conseguido reducir el error en un valor superior a este parámetro durante la última iteración. Es, por tanto, el umbral de convergencia del algoritmo

Secundariamente se expondrán los diferentes bloques en los que se puede dividir este algoritmo, por lo que el resto de variables contenidas en *MSBCv* se expondrán durante este proceso.

El algoritmo se puede dividir en 3 bloques o fases diferentes, que se pueden resumir como:

1. **Inicio del algoritmo.** Comienza el algoritmo cargándose únicamente un *clúster* que aglutinará a todas las muestras.

<sup>7</sup>Se debe entender por K-medias básico el algoritmo que se basa en dado un conjunto de *K clusters* cuyos *centroides* tienen una posición predefinida, encontrar la distribución de esos *K clusters* que hace mínimo el error cuadrático medio

<sup>8</sup>El bucle de bisección incluirá al algoritmo de K-medias básico, esto es, una vez que se fija el nuevo número de *clusters*, se realiza K-medias básico para encontrar la mejor distribución para ese número de *clusters*

2. **Bucle de bisección de *clusters*.** Se ejecuta un bucle que se encargará de ir dividiendo los *clusters*. Cada vez que se divida algún *clúster* o todos los *clusters*, en función de la modalidad, se ejecutará el algoritmo básico de K-medias.

Este bucle finalizará cuando se alcance un número prefijado de *clusters*, *MSBCv.clusters\_required*, o el aumento de *clusters* no reduzca el error en al menos un valor fijado por *MSBC.minimun\_error\_recoverable*.

3. **Algoritmo básico de K-medias.** Dentro del bucle anterior, con una cantidad de *clusters* definida, se ejecutará otro bucle en el que se buscará reducir el error para ese número de *clusters*.

Este bucle finalizará si en una iteración el error no se ha reducido en un valor superior al fijado por la variable mencionada anteriormente, *MSBCv.minimun\_error\_recoverable*.

Estos bloques hacen uso de un conjunto de procesos elementales como son el cálculo de *centroides* o la estimación del error cuadrático. Estos se describen a continuación.

#### 4.6.3.2. Procesos elementales

Cabe recordar que el espacio muestral se corresponde con los tamaños de mensaje almacenados para una aplicación,  $n$ , en una posición de secuencia de mensaje,  $l$ , y para una dirección *upstream* o *downstream*, por lo que cada *clúster* se identifica con  $C_{k,l,n}$  siendo  $k$  el identificador del *clúster* dentro del vector *clusters*.

No obstante, durante la exposición de los diferentes procesos elementales será ese último identificador,  $k$ , el único que se explicitará pues se pretende solamente diferenciarlo del resto *clusters* para el modelado de ese espacio muestral.

##### 4.6.3.2.1. Asignación de pertenencia

Localizar cual es el *clúster* cuyo *centroide* es el más cercano a cada muestra es una de las principales tareas del algoritmo. Para ello, se deben analizar todas las muestras, estimando cual es el *centroide*, de los *clusters* activos en el proceso (variable *n\_clusters\_active*), más cercano.

El resultado de este proceso será almacenado en el vector *array\_belong*, donde cada posición del vector se corresponde con una muestra y el contenido de esa posición con el identificador,  $k$ , del *clúster* al que esta asignada la muestra.

Los pasos seguidos quedan esquematizados en el diagrama de flujo de la Figura 4.20 que se pueden resumir en lo siguiente:

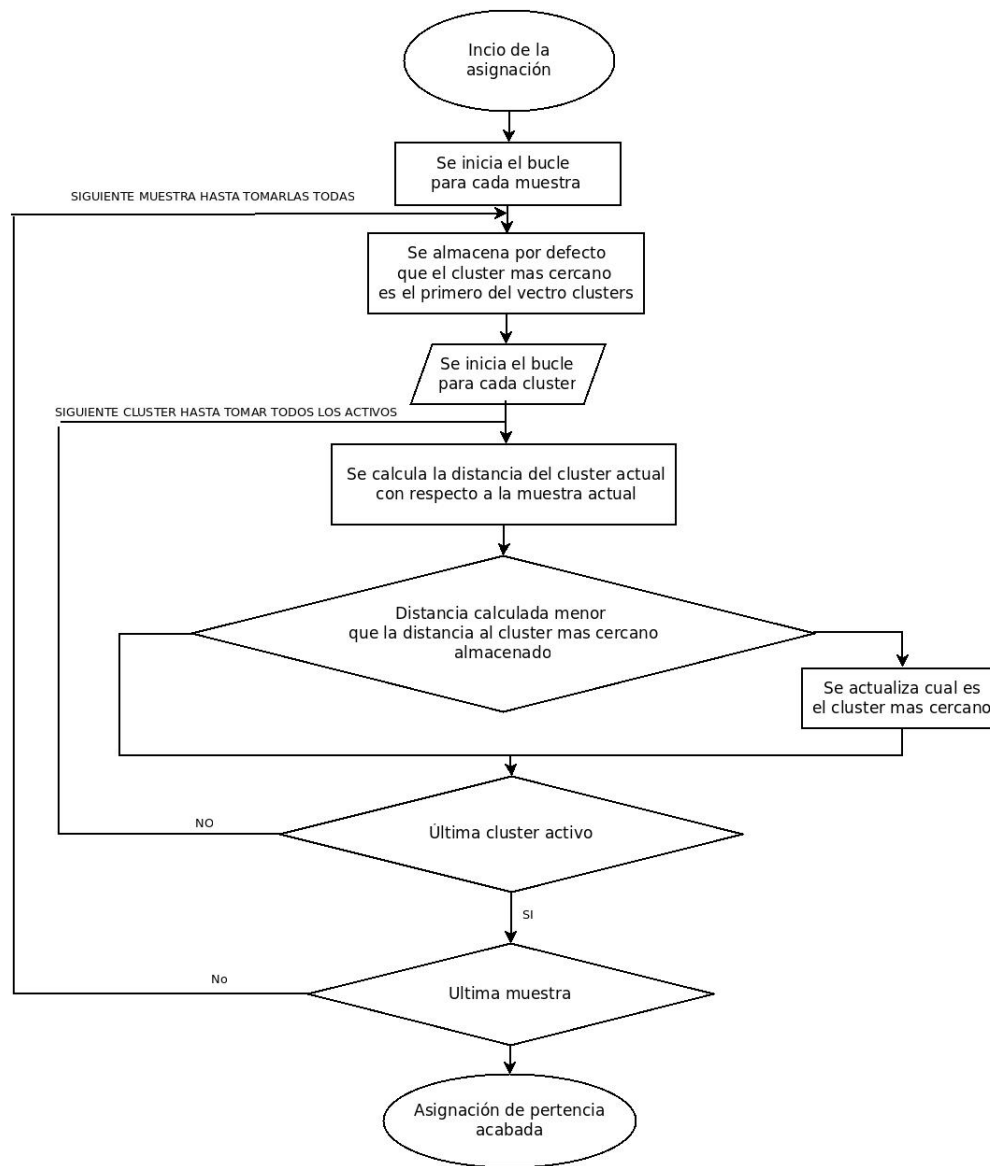


Figura 4.20: Arquitectura del procedimiento de asignación de cada muestras al *clúster* con el *centroide* más cercano

- Se crea un bucle que recorra cada una de las muestras,  $x_i$ . Para cada muestra se analizará cuál es el *clúster* al que pertenece. Para ello, se usará la distancia al *centroide* correspondiente para cada *clúster*, seleccionándose el más cercano.
- Inicialmente se toma el primer *clúster* del vector de *clusters*, es decir  $C_0$ , como el *clúster* al que pertenece la muestra, con objeto de inicializar la variable  $C_{cercano}$  que se utilizará como parámetro para

almacenar la posición el *clúster* con el *centroide* más cercano para la muestra que se está analizando en esa iteración.

- Se crea otro bucle dentro del anterior que recorrerá los demás *clusters* activos <sup>9</sup>. Se irá comprobando si la distancia entre la muestra  $x_i$  y el *centroide*  $\zeta_k$  del *clúster* tomado por el bucle en esa repetición, es menor que la distancia entre la muestra y el *clúster* que se supone cercano con *centroide*  $\zeta_{\text{cercano}}$ .

La distancia se calculará con la fórmula:

$$D(x_i, \zeta_k) = \left( \frac{\zeta_k}{B + |\zeta_k|} - \frac{x_i}{B + |x_i|} \right)^2$$

- En caso de que la distancia estimada con respecto a ese *centroide* sea menor, se almacenará el índice de posición de dicho *clúster* registrándose en la variable  $C_{\text{cercano}}$ .
- Cuando finalice el bucle que toma cada *clúster*, se tendrá la posición del *clúster* más cercano a la muestra analizada en  $C_{\text{cercano}}$ , esto se almacenará en el vector *array\_belong* en la posición dada por la muestra que se esta tratando. Se repetirá el proceso para las siguientes muestras hasta que se hayan recorrido todas.

Durante la fase de inicio del algoritmo este proceso no se lleva a cabo ya que todas las muestras pertenecerán al único *clúster* que hay.

#### 4.6.3.2.2. Cálculo de *centroides*

Esta tarea se encarga de, una vez definidas qué muestras están en cada *clúster*,  $C_k$ , redefinir el *centroide*  $\zeta_k$  de cada cluster. Se permiten dos métodos para estimar este *centroide* en función de la variable *MSBCv.euclidean\_mean* que son los siguientes:

- Aplicando la distancia euclídea:

$$D(x_i, \zeta_k) = |x_i - \zeta_k|$$

- Aplicando la distancia euclídea escalada:

$$D(x_i, \zeta_k) = \left| \frac{x_i}{|x_i| + B} - \frac{\zeta_k}{|\zeta_k| + B} \right|$$

Como se detalló en el Apartado 3.4.2, el proceso es similar para ambos modos, ya que se hará la media a las muestras o a su valor escalado. Por ello, solamente variará el principio y el final de la evaluación, donde se tendrá que escalar y de-escalar.

Cabe mencionar que se tomará el valor absoluto de las muestras para posteriormente asignar el signo en función de la dirección de los mensajes.

La tarea esquematizada en la Figura 4.21 tiene los siguientes pasos:

<sup>9</sup>Resto de *clusters* activos en ese ejecución. Al principio se tiene únicamente un *clúster*, aumentando progresivamente el número de *clusters* activos.

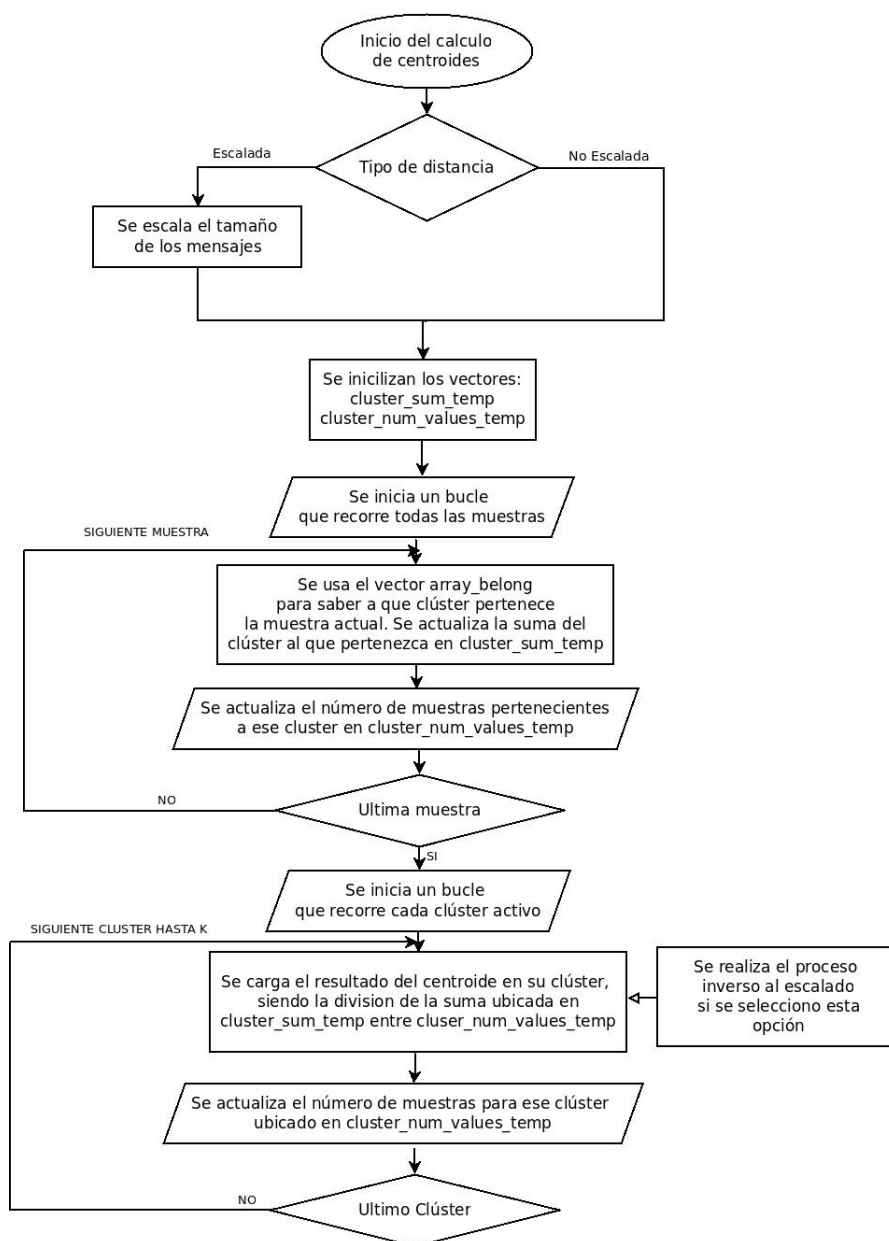


Figura 4.21: Estructura del procedimiento empelado para el cálculo del *centroide* de un *clúster* dado

- Se inicializan a cero dos vectores:
  - Vector *cluster\_sum\_temp*. Almacena la suma temporal de todas las muestras que pertenezca a cada *clúster*. Cada posición del vector se corresponderá con el *clúster* que ocupe esa misma posición en el vector de estructuras *clusters*, ver Figura 4.19.

- Vector *cluster\_num\_values\_temp*. Almacena el número de elementos que contiene cada *clúster*. De nuevo, cada posición de este vector estará relacionada con el *clúster* que ocupa la misma posición en el vector de estructuras antes mencionado.
- Hecho esto se escalarán los valores de las muestras escaladas si se ha seleccionado esa opción:

$$\chi_i = \frac{x_i}{|x_i| + B}$$

- Se obtiene el *centroide* como la de una media aritmética, tal que:

$$\zeta_k = \frac{1}{N_k} \sum_{x_i \in C_k} x_i$$

o, para muestras escaladas:

$$\vartheta_k = \frac{1}{N_k} \sum_{\chi_i \in C_k} \chi_i$$

por lo que se usará el vector *cluster\_sum\_values\_temp* para contabilizar esta sumatoria. Para ello, se creará un bucle que recorra todas las muestras que, ayudado por el vector *array\_belong*, sabrá a qué *clúster* está asignada cada muestra y por lo tanto su posición en el vector *cluster\_sum\_values\_temp*, acumulará su valor con el resto de la suma para cada *clúster* ya contenida en el vector antes mencionado.

Durante este bucle también se actualizará el vector *cluster\_num\_values\_temp*, contabilizando el número de muestras para cada *clúster*,  $N_k$ .

- Finalizado el bucle anterior, se procederá con el siguiente bucle que tomará en cada iteración uno de los *clusters* calculando su media, es decir, dividiendo su valor *cluster\_sum\_temp* entre *cluster\_num\_values\_temp*.

Obtenida esa media se deberá de-escalar, si se ha seleccionado esa opción:

$$\zeta_{k,abs} = \frac{B * \vartheta_{k,abs}}{1 - \vartheta_{k,abs}}$$

- Finalmente, ya se tiene el *centroide*  $\zeta_k$ , en valor absoluto, por lo que se le asignará el signo en función de la dirección. La dirección se puede comprobar en las muestras usadas para calcular el *centroide*, observando el signo.



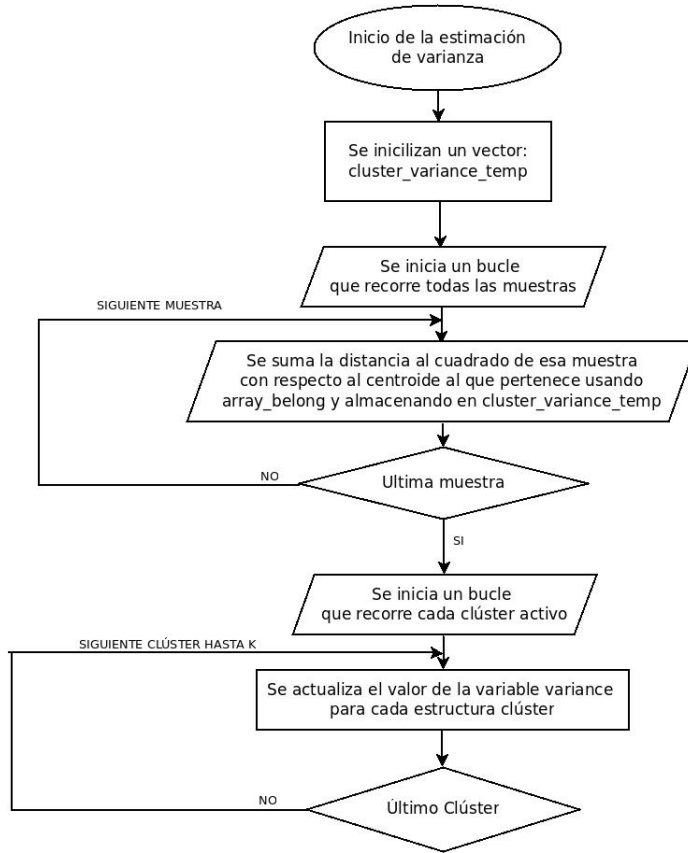


Figura 4.22: Estructura del proceso de estimar la varianza para los distintos *clusters*

#### 4.6.3.2.3. Cálculo de la varianza

Una vez evaluado el *centroide*,  $\zeta_k$ , de cada *clúster*,  $C_k$ , se procederá a estimar el error asociado al modelado actual basado en *clusters*. Para ello, el primer paso es conocer la varianza,  $\sigma_k^2$ , de cada *clúster*, ya que el error cuadrático total,  $\xi$ , es la suma pesada de la varianza de todos los *clusters* en el modelo tal que:

$$\xi = \sum_{k=1}^K w_k * \sigma_k^2$$

la varianza  $\sigma_k^2$  para cada *clúster*  $C_k$

$$\sigma_k^2 = \sum_{x_i \in C_k} |D(x_i, \zeta_k)|^2$$

Este proceso es similar al anterior, pues en primer lugar, deberemos recorrer las muestras, generando la sumatoria, y posteriormente cada uno de los diferentes *clusters* para asignar sus respectivas varianzas. Un esquema de esta tarea se muestra en la Figura 4.22.

La secuencia de acciones llevadas a cabo se pueden resumir en:

- De nuevo, es necesario un vector de longitud igual al número de *clusters* activos, que se llamará *cluster\_variance\_temp*.

En este vector se almacenará la sumatoria de la varianza para cada uno de los *clusters*,  $\sum_{k=1}^K \sigma_k^2$ , previamente a ser dividido por el número de muestras que pertenecen a ese cluster,  $N_k$ .

Cada posición del vector definido anteriormente se corresponde con el *clúster* que ocupa la misma posición en el vector de estructuras llamado *clusters*.

- Un bucle tomará en cada iteración cada una de las muestras,  $x_i$ , y calculará la distancia a su respectivo *centroide*,  $\xi_k$ ,  $D(x_i, \xi_k)$ , elevándola posteriormente al cuadrado y sumándola en la posición,  $k$ , de su *clúster* en el vector *cluster\_variance\_temp*.
- Finalizado el bucle anterior, se cargará el valor de la varianza de cada *clúster*, una vez dividido por el número de muestras de cada *clúster*  $N_k$ , en la variable, llamada *variance*, habilitada para ello en la estructura de tipo *cluster*, véase la Figura 4.19. Para ello, es necesario otro bucle que recorra cada uno de los *clusters* activos.

#### 4.6.3.2.4. Cálculo del error cuadrático medio

Esta tarea es útil a la hora de evaluar la calidad del modelo de *clusters* estimado, ya que determina cuando se debe de parar el proceso de bisección de *clusters*, o el proceso de reducción de errores para un número de *clusters* fijo, K-medias básico.

Como se comentó en la sección anterior, el error cuadrático  $\xi$  se calculará a partir de la varianza de cada uno de los *clusters* del modelo. Cada *clúster* tendrá definida su varianza dentro de su estructura de tipo *cluster*.

Por lo tanto, únicamente se procederá a recorrer este vector de estructuras realizando la suma ponderada de la varianza  $\sigma_k^2$  de cada *clúster*

$$\xi = \sum_{k=1}^K w * \sigma_k^2$$

Los pasos seguidos se esquematizan en la Figura 4.23:

- Se define una variable para almacenar el valor acumulado de la sumatoria que se llamará *estimate\_error*.
- Se genera un bucle que recorra cada una de esas posiciones del vector de estructuras tipo *cluster*, tomándose en cada iteración el valor de la variable *variance* de cada una de esas estructuras, multiplicándolo por su peso relativo y acumulando el resultado a la variable *estimate\_error*.  
Previamente a la suma se comprobará que ninguno de estos *clusters* se ha quedado vacío, evaluando la variable *num\_values\_content*, presente en toda estructura *cluster*, véase la Figura 4.19.

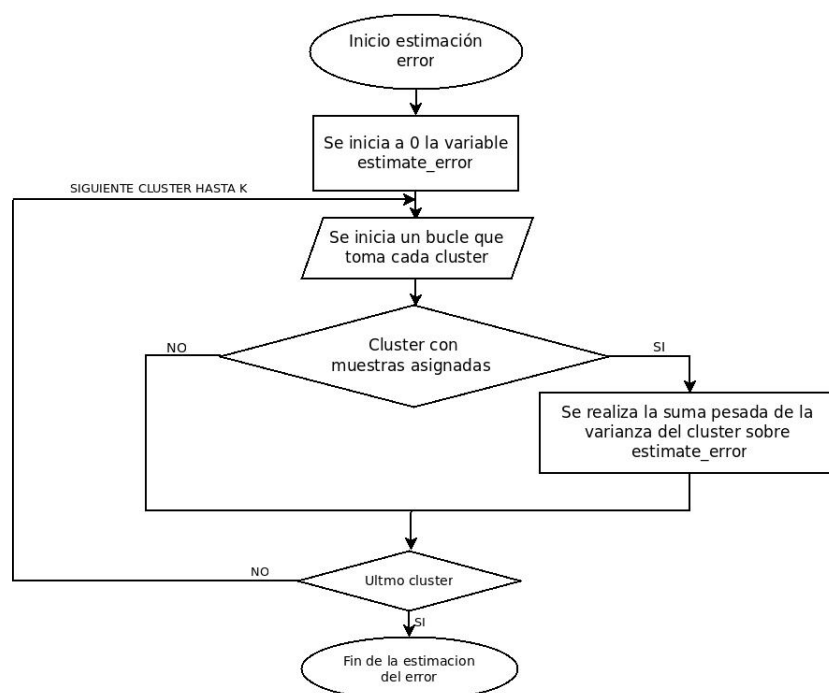


Figura 4.23: Estructura del procedimiento seguido para el cálculo del error cuadrático medio total

El peso de cada *clúster*,  $w_k$ , vendrá dado por el número de muestras que están asignadas al clúster,  $N_k$ , y por el total de muestras disponibles en el modelado,  $N$ .

$$w_k = \frac{N_k}{N}$$

- La tarea terminará cuando finalice el bucle, asignándose la nueva estimación de error a cualquiera de las variables dispuestas para ello como *estimated\_error*, *last\_estimated\_error* o *up\_last\_estimated\_error*.

#### 4.6.3.2.5. Bisección de los clusteres

Esta tarea se encarga, de dado un *clúster*, dividirlo en otros dos *clusters*. Como ya se explicó en el Apartado 3.4.3, se pueden usar dos modos de llevar a cabo esta bisección: globalmente a todos los *clusters* activos o solamente al *clúster* con más varianza.

De nuevo, el concepto de distancia empleado dará lugar a dos posibles implementaciones: bien usando la distancia euclídea o usando la distancia definida a partir de la función de escalado. Esto se podrá determinar desde un fichero externo al programa, a partir de la variable *MSBCv. euclidean\_shiftf*.

Los pasos que se siguen para cada uno de estos dos modos son:

- Modelo con distancia euclídea:
  - Los dos *centroides* creados, tanto el generado con desplazamiento positivo  $\zeta_{k,+}$  como el generado con desplazamiento negativo  $\zeta_{k,-}$ , estarán a la misma distancia del *centroide* referencia,  $\zeta_k$ , por lo que hará falta aumentar o disminuir el valor absoluto de ese tamaño del mensaje.  
El total de desplazamiento vendrá determinado por una variable externa llamada *MSBCv.fixed\_shift*. Por lo tanto, la expresión resultante es:
 
$$\begin{cases} |\zeta_{k,+}| = |\zeta_k| + MSBCv.fixed\_shift \\ |\zeta_{k,-}| = |\zeta_k| - MSBCv.fixed\_shift \end{cases}$$
  - Después se asigna el signo a ambos *clusters* en función de la dirección de los mensajes, que se puede obtener del signo de las muestras.
- El proceso a seguir si se aplica la distancia escalada fue expuesto en el Apartado 3.4.3. En la implementación, lo que se hace es simplemente aplicar la expresión 3.31 que, tomando valor absoluto para cualquier centroide,  $\zeta$ , resulta:

$$\begin{cases} |\zeta_{k,+}| > |\zeta_k| & |C_{\zeta,+}| = \frac{|\zeta_k| + \eta\sigma(B + |\zeta_k|)}{1 - \sigma\eta(1 + \frac{|\zeta_k|}{B})} \\ |\zeta_{k,-}| < |\zeta_k| & |\zeta_{k,-}| = \frac{|\zeta_k| - \eta\sigma(B + |\zeta_k|)}{1 + \sigma\eta(1 + \frac{|\zeta_k|}{B})} \end{cases}$$

La variable  $\eta$  modula el desplazamiento y puede ser definida también desde fuera del código del programa. La variable encargada de esto es *MSBCv.step\_modifier*

- Cómo en el proceso anterior, se deberá asignar signo en función de la dirección de los mensajes.

#### 4.6.3.3. Inicio del algoritmo

En esta fase inicial se deberán realizar muchos de los procesos que serán repetidos durante las dos siguientes fases. Algunos de estos procesos son el cálculo del • para un *clúster* o el cálculo del error cuadrático medio para el modelado actual.

Sin embargo, la complejidad de estos procesos es menor a la que se tendrá durante la siguiente fase, ya que al comienzo del algoritmo todas las

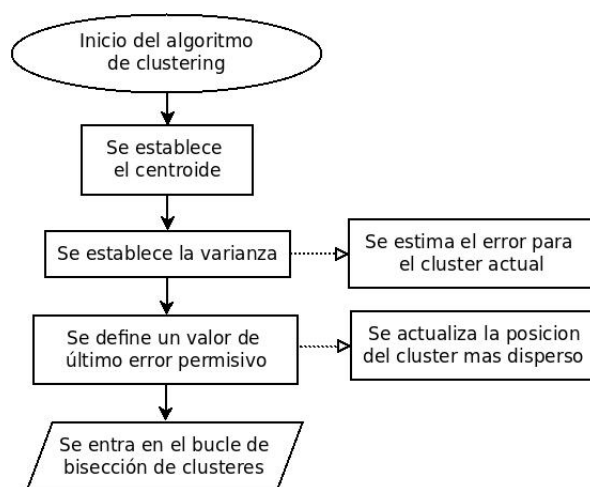


Figura 4.24: Inicio del algoritmo de *clustering*

muestras pertenecen al mismo cluster, por lo que no se necesita la reasignación de muestras.

En la Figura 4.24 se muestra el orden en el que se ejecutan los diferentes pasos que se enumerarán seguidamente, pero sin entrar en detalles de su implementación la cual ya fue expuesta en las secciones anteriores.

Los procedimientos realizados son los siguientes:

1. Se comienza estimando el *centroide*. Este es el primer paso, ya que se necesita calcular una posición inicial para este elemento y, a partir de ahí, comenzar con la bisección, consiguiéndose así que los futuros *clusters* estén lo mejor ubicados que se pueda.
2. Como se verá en la siguiente fase, una de las condiciones restrictivas para que el bucle de la bisección pueda repetirse es que el valor del error se haya reducido en la última iteración. Por lo tanto, es necesario calcular, previamente a la primera repetición, el valor del error para que se consiga entrar en el bucle.
3. Durante la exposición de las distintas variables presentes en el sistema se mencionó el parámetro *estimated\_error*, que permite almacenar el error actual, así como la variable *up\_last\_estimate\_error* que será la que se encargue de almacenar el error previo a la repetición.

Por ello, para que el bucle arranque se cargará en esta variable un valor de error que será el de *estimated\_error* mas el doble de la mínima reducción de error necesaria, dada por el parametro global *MSBCv.minimun\_error\_recoverable*, para que el bucle comience,

$$up\_last\_estimated\_error = estimated\_error + 2 * MSBCv.minimun\_error\_recoverable$$

4. El cálculo del error hace necesario que se calcule la varianza, ya que como se expuso en el Apartado 4.6.3.2.4 el error cuadrático medio se calcula a partir de la varianza de cada *clúster*.

Además, otra de las condiciones del bucle obliga a que el *clúster* con más varianza tenga un valor mínimo de esto dado por la variable global *MSBCv.min\_variance*. El propósito de este conjunto de restricciones se explicará en el siguiente apartado.

#### 4.6.3.4. Bucle de bisección de clusteres

Iniciado el proceso con las acciones anteriores, inmediatamente se entrará en el bucle de bisección de *clusters*, en el cual se realizarán las tareas pertinentes para poder llevar a cabo una división de *clusters* efectiva. Estas tareas se esquematizan en la Figura 4.25.

Este bucle tendrá un número indeterminado de iteraciones, debido a que es desconocido cuando alguna de las condiciones dejará de cumplirse y, por lo tanto, el procedimiento ha dejado de ser efectivo, alcanzándose el mejor modelado posible para el conjunto de muestras en cuestión.

Las restricciones de repetición planteadas para este bucle son las siguientes:

- El número de *clusters* activos, indicado por la variable *n\_cluster\_required*, deberá ser inferior al tope fijado desde la variable global *MSBCv.clusters\_active*.

$$n\_clusters\_active < MSBCv.clusters\_required$$

En principio este número de *clusters* máximo debe ser elevado permitiéndose definir todos los clusteres necesarios, ya que como se expuso en el Apartado 3.3.1, cada *clúster* nace con el objetivo de definir un método dentro de la aplicación. No obstante, se limita el número ya que se pretende evitar que el proceso se alargue indefinidamente o que salgan tantos *clusters* que hagan que el modelo no sea representativo de la aplicación.

- El *clúster* con más varianza, cuya posición en el vector de estructuras de tipo *cluster* viene dado por *most\_dispersed\_cluster\_pos*, debe tener al menos una varianza superior a un valor mínimo establecido por una variable global *MSBCv.min\_variance*.

$$clusters[most\_dispersed\_cluster\_pos].variance > MSBCv.min\_variance$$

Con esta condición se pretende evitar que se continúe con el bucle si la varianza presente en los *clusters* es muy baja, siendo el más disperso el tomado como indicador de esto, ya que apenas se conseguirá mejorar el modelado actual.

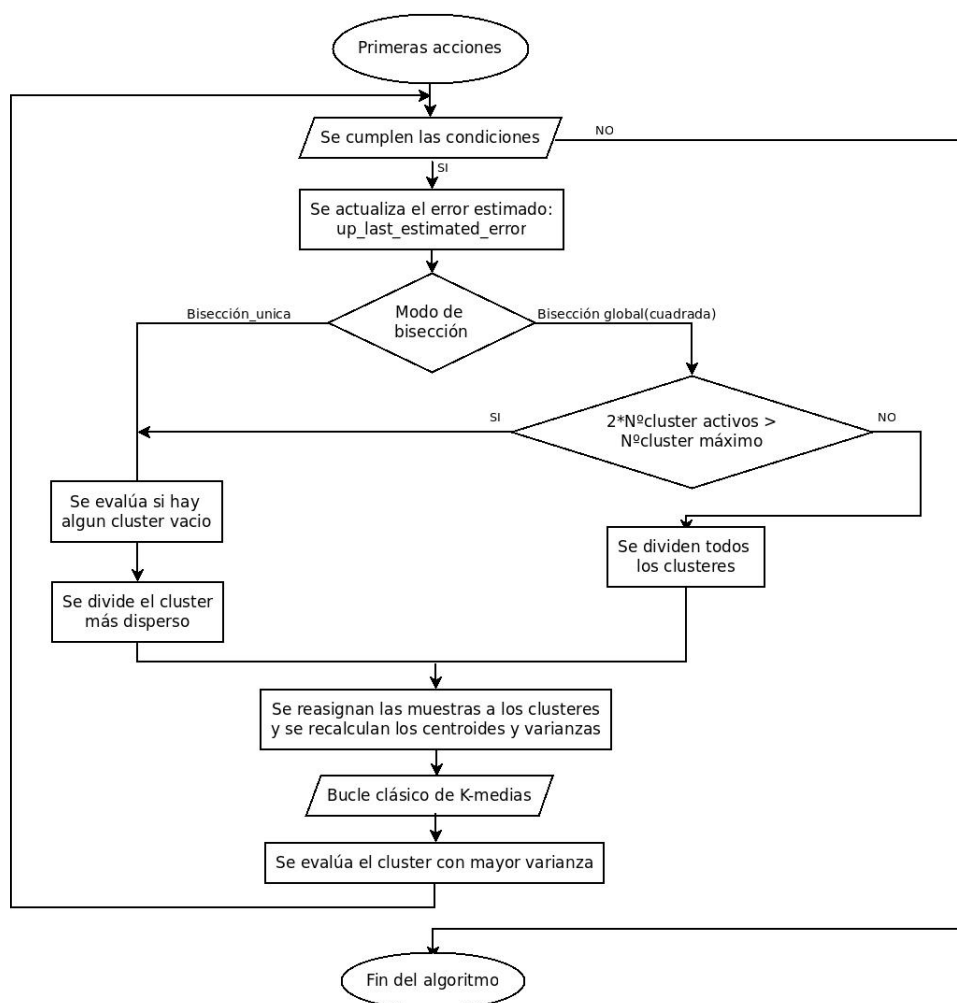


Figura 4.25: Bucle de bisección de clústeres

- Por último, la restricción sobre el error cuadrático medio recuperado en la última iteración. Como ya se ha mencionado en varias ocasiones, existe una variable que almacena el valor de la estimación del error previo a una iteración, que se denomina *up\_last\_estimated\_error*.
- Cuando una iteración concluye, se comprueba si el error actual, almacenado en *estimated\_error*, es suficientemente inferior al valor del error que se tenía antes de la repetición almacenada en la variable mencionado en el párrafo anterior. El suficientemente inferior viene definido por la variable global *MSBCv. minimum\_error\_recoverable*:

$$up\_last\_estimated\_error > MSBCv. minimum\_error\_recoverable + estimated\_error$$

La implementación de la segunda condición, la que versa sobre la mínima varianza para los *clusters*, es discutible ya que, en principio, es posible que el modelado converja sin tener en cuenta esa restricción. Sin embargo, experimentalmente se reducía el número de iteraciones de forma considerable por lo que se decidió incluirla.

Enumeradas las condiciones, queda por explicar el conjunto de tareas que se llevan a cabo en cada una de las diferentes repeticiones del bucle. Se pueden resumir en lo siguiente:

1. Se comprueba que se cumplen la serie de restricciones antes expuestas.
2. Se actualiza el valor de *up\_last\_estimated\_error* con el valor del error cuadrático medio, antes de que se realice la bisección y el posterior algoritmo de K-medias.
3. Se evalúa el modo de bisección elegido para la ejecución en cuestión. La elección de un modo de ejecución u otro viene determinado por una variable global llamada *MSBCv.kmeans\_model*.

Los dos posibles modos son:

- **Bisección del *clúster* con más varianza:** Solamente se dividirá aquel *clúster* que este más disperso, es decir, con la mayor varianza.

Este proceso hace que se aumente en un *clúster* el modelado actual. Se ha diseñado el modelo para que el nuevo *clúster* pueda ocupar el espacio de alguno de los *clústeres* ya creados, es decir, que tienen posición en el vector de *clústeres*, véase la Figura 4.19, pero que se han quedado vacíos durante las diferentes repeticiones.

- **Bisección de todos los *clústeres*:** Todos los *clústeres* serán divididos.

Con este procedimiento, se duplicará el número de *clusters* activos, indicado por *n\_clusters\_activos*. Si el doble de el número de *clusters* actual es mayor que el máximo fijado por *MSBCv.clusters\_required*, se ejecutará por defecto el otro de método de bisección durante el resto de repeticiones.

4. Realizada la división pertinente de los *clusters* del modelo, se continuará recalculando los diferentes parámetros característicos de cada *clúster*. Para ello se ejecutan los siguientes procesos:
  - a) Se reasignan las muestras a los *clusters* cuyos *centroides* estén mas cercanos.
  - b) Se reestiman los *centroides* de los diferentes *clusters*.
  - c) Se recalcula la varianza para cada *clusters*.



d) Se estima de nuevo el error.

5. Se actualiza el valor de la variable *last\_estimated\_error*, cuyo fin es similar al de la variable *up\_last\_estimated\_error*, pero esta vez con respecto al bucle del algoritmo básico de K-medias.

La variable antes mencionada será cargada con un valor que permita que se pueda entrar en el siguiente bucle, como es el doble del error mínimo que se ha de reducir en cada iteración:

$$last\_estimated\_error = 2 * MSBCv. minimum\_error\_recoverable + estimated\_error$$

6. Hecho esto, se entrará en el bucle siguiente que será explicado en el siguiente capítulo.
7. Cuando el algoritmo de K-medias básico contenido dentro de este bucle termine, se actualizará cual es el *clúster* con mayor dispersión, con objeto de que se evalúe en las restricciones de la siguiente iteración.

#### 4.6.3.5. Bucle del algoritmo clásico K-medias

Este segundo bucle, contenido dentro del que se ha expuesto en la sección anterior, tiene como finalidad obtener el mejor modelado posible para un número fijo de *clusters*.

Al ejecutarse una repetición del bucle de bisección se aumentará el número de *clusters*. Con este algoritmo se pretende que, dado el número de *clusters* por el otro bucle, ajustar al máximo el error cuadrático medio.

De nuevo, tenemos un conjunto de condiciones para que el bucle itere. En esta caso son las siguientes:

- Error mínimo recuperado en la última repetición. Esta condición es similar a una de las expuestas para el otro bucle, pues hace uso de la variable global *MSBCv. minimum\_error\_recoverable* para determinar si se ha reducido suficientemente el error.

La variable que contiene el error cuadrático medio antes de la repetición es *last\_estimated\_error*, por lo que la expresión de esta condición es la siguiente:

$$last\_estimated\_error > MSBCv. minimum\_error\_recoverable + estimated\_error$$

- Otra variable adicional con respecto al error que se ha añadido es que el error ha de ser superior a ese error mínimo recuperable fijado para la ejecución tal que:

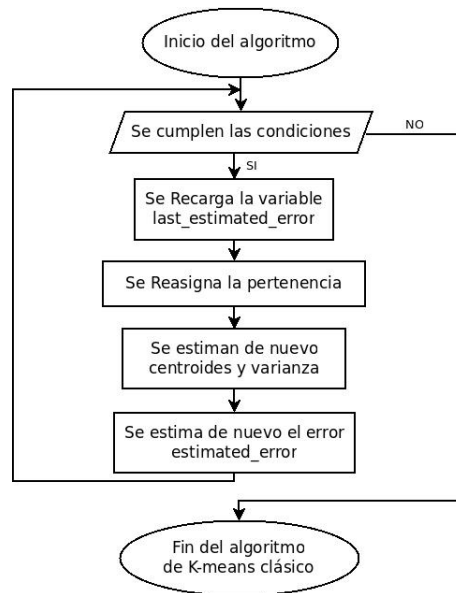


Figura 4.26: Algoritmo K-medias básico

$$estimated\_error > MSBCv. \ minimum\_error\_recoverable$$

Los pasos necesarios para completar este algoritmo son los propios del K-medias básico ( Figura 4.26), que se pueden resumir en los siguientes:

1. Antes de llevarse a cabo la ejecución de esas tareas, se ha de almacenar el error previo a la iteración en la variable *last\_estimated\_error*.
2. Se reasignarán las muestras.
3. Se reestimarán el *centroide* de cada clúster.
4. Se recalculará la varianza de cada *clúster*.
5. Se reestimarán el error cuadrático medio.
6. Se repiten todos los pasos anteriores si durante la última iteración se ha conseguido reducir el error en un valor superior a el establecido por *MSBCv. minimum\_error\_recoverable*. Para ello, se compara el error almacenado en *last\_estimated\_error* con el error actual.

#### 4.6.4. Acciones finales del entrenamiento

Una vez que se hayan entrenado los diferentes modelos para esa aplicación,  $n$ , esto es, todos los modelos para todos los mensajes de la secuencia,  $l$ , y para ambas direcciones, se imprimirán los resultados en el archivo de salida del entrenamiento, denominado *own\_MSBC\_signature\_file.txt*.

Cabe recordar, que se habían pre-cargado todas las muestras para una aplicación en la estructura que se puede observar en la Figura 4.16. Consecuentemente, no se continuará con la siguiente aplicación hasta que se disponga de un modelo para todos los conjuntos de muestras de esa aplicación, es decir, para cada posición de la secuencia y para ambas direcciones.



## Capítulo 5

# Pruebas y simulaciones

En este capítulo se describen algunas de las pruebas y simulaciones llevadas a cabo con el objetivo de comprobar el correcto funcionamiento del sistema. Algunas de las capacidades del sistema que van a ser evaluadas son:

- La capacidad de extraer correctamente la secuencia de mensajes de un flujo dado.
- La capacidad de generar un modelado basado en gaussianas multiplico, dado el conjunto de muestras.
- Finalmente, la capacidad de clasificar tráfico dados una serie de modelos.

Se comenzará con una sencilla prueba de un único flujo en la que se comprobará cómo el sistema es capaz de extraer la secuencia de mensajes de esa sesión de forma correcta.

### 5.1. Extracción de parámetros para un flujo SMTP

El protocolo SMTP (*Simple Mail Transfer Protocol*) es utilizado en el intercambio de correo electrónico. Es un protocolo clásico y ampliamente conocido, por lo que será utilizado para esta primera prueba.

Se usará un flujo SMTP, cuyo intercambio de paquetes, analizado con *Wireshark*, es mostrado en la Figura 5.1. Estos paquetes se usarán como tráfico de entrada del programa, evaluando si el sistema de extracción de parámetros funciona correctamente. Este flujo ha sido descargado de una página web [17], desde donde se puede acceder a diversos ejemplos de tramas intercambiadas por aplicaciones, contenidas en archivos en formato *libpcap*.

Previamente, se pasará este mismo flujo por el *plugin plugin l7\_1.3* para que se genere un archivo de pre-clasificación, *ground\_file.tie*, ya que para un intercambio SMTP este clasificador si que es eficaz debido a que, como

Filter:		tcp.port == 25		▼	Expression...	Clear	Apply	Guardar
No.	Time	Source	Destination	Protocol	TCP payload length	L		
3	0.036986	10.10.1.4	74.53.140.153	TCP	1) Handshaking	0		
4	0.383936	74.53.140.153	10.10.1.4	TCP		0		
5	0.383968	10.10.1.4	74.53.140.153	TCP		0		
6	0.727603	74.53.140.153	10.10.1.4	SMTP	2) Intercambio SMTP	181		
7	0.732749	10.10.1.4	74.53.140.153	SMTP		9		
8	1.073326	74.53.140.153	10.10.1.4	TCP		0		
9	1.074123	74.53.140.153	10.10.1.4	SMTP		137		
10	1.076669	10.10.1.4	74.53.140.153	SMTP		12		
11	1.419021	74.53.140.153	10.10.1.4	SMTP		18		
12	1.419595	10.10.1.4	74.53.140.153	SMTP	30			
13	1.761484	74.53.140.153	10.10.1.4	SMTP	18			

Figura 5.1: Intercambio de tramas para un flujo SMTP

antes se ha mencionado, SMTP es ampliamente conocido y cuenta con una estructura bien conocida. Esto se hace con la siguiente orden:

```
./TIE -m o -r smtp.pcap -a l7_1.3 -E ground_file -S 1000
tcp.port 25
```

La finalidad de cada uno de los argumentos usados en la orden viene expuesta en la Tabla 5.1. El resultado es el fichero *ground\_file.tie*, que se muestra en la Figura 5.2.

En esta figura se comprueba que el proceso de clasificación es exitoso, pues se le ha asignado la *app\_id* 15, que se corresponde con SMTP, véase el archivo de asociación de identidad de TIE, *TIE.apps.txt*. Con estos pasos se ha conseguido contar con un archivo de pre-clasificación. Seguidamente

Comandos de TIE	
Argumento	Utilidad
<b>-m o</b>	Indica que el modo de ejecución será <i>offline</i>
<b>-r smtp.pcap</b>	Indica al archivo smtp.pcap como entrada del programa
<b>-a l7_1.3</b>	Indica que el clasificador l7_1.3 será el usado en la ejecución
<b>-E ground_file</b>	El archivo <i>ground_file.tie</i> almacenará el resultado de la clasificación
<b>-S 1000</b>	Almacenará los 1000 primeros bytes de carga útil de cada sesión, necesario para el plugin <i>l7_1.3</i> , ya que necesita analizar los primeros bytes intercambiados en busca de firmas
<b>port 25</b>	Solo se analizarán los flujos al puerto 25, puerto correspondiente a la aplicación SMTP

Tabla 5.1: Comandos usados para pre-clasificación con TIE al aplicar el clasificador l7\_1.3

```

ground_file.tie x
# TIE output version: 1.0 (text format)
# generated by: . -m o -r smtp.pcap -a l7_1.3 -E ground_file -S 1000 port 25

# Working Mode: off-line
# Session Type: biflow
# 1 plugins enabled: l7

# begin trace interval: 1254722767.529046

# begin TIE Table
# id      src_ip      dst_ip      proto  sport  dport  dwpkts  uppkts  dwbytes  upbytes
# t_start t_last
0         10.10.1.4     74.53.140.153  6      1470   25     7        7       384      105
1254722767.529046  1254722775.106759  15      0       100
# end of text table

```

Figura 5.2: Fichero *ground\_file.tie* generado por la pre-clasificación de un flujo SMTP

se procederá a analizar la parametrización para nuestro clasificador con el comando:

```

./TIE -m o -r smtp.pcap -a MSBC_1.1 -e /output/ground_file.tie
-l 20 -u 512 -U 5 port 25

```

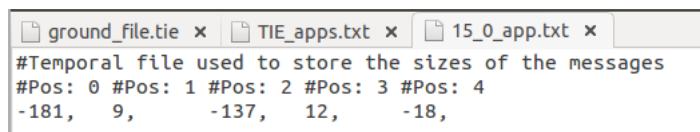
Los argumentos utilizados son similares a los de la orden anterior, excepto los propios del *plugin* implementado, mostrados en la Tabla 5.2. Como se explicó en el Apartado 4.6, cada aplicación generará su propio archivo en el que se irán almacenando las distintas muestras para cada uno de los mensajes de la secuencia.

En el archivo *15\_0\_app.txt*, que se muestra en la Figura 5.3, se almacenarán las muestras de tamaño de mensaje para la aplicación SMTP. El flujo analizado sigue el esquema de intercambio analizado en la Figura 5.1, donde se puede comprobar que los tamaños de mensaje que hay expuestos para cada intercambio de paquetes son los almacenados posteriormente. Si se analiza detenidamente, se puede verificar que:

1. Los primeros tres mensajes corresponden al "handshake" de TCP, por lo que no contienen ningún tipo de dato SMTP, consecuentemente, no siendo almacenados.

Argumentos usados por el clasificador	
Comando	Utilidad
<b>-u 512</b>	Establece en 512 bytes el mínimo tamaño de la carga útil de un paquete para que no se considere que la evaluación del mensaje ha concluido
<b>-U 5</b>	Indica el número de mensajes que se han de extraer por flujo

Tabla 5.2: Argumentos usados para el entrenamiento del clasificador propuesto en un flujo SMTP



```

ground_file.tie x TIE_apps.txt x 15_0_app.txt x
#Temporal file used to store the sizes of the messages
#Pos: 0 #Pos: 1 #Pos: 2 #Pos: 3 #Pos: 4
-181, 9, -137, 12, -18,

```

Figura 5.3: Almacenamiento de muestras para un flujo SMTP

2. Como la comunicación la inicia el cliente durante el acuerdo de conexión, el primer mensaje es considerado *downstream*, signo negativo, ya que va dirigido del servidor al cliente, que es el sentido contrario con el que se comenzó la comunicación.
3. Como la carga útil de los paquetes intercambiados es inferior al mínimo de bytes definido en los argumentos de la orden de ejecución de TIE, la carga útil de cada paquete será un mensaje en sí mismo. Los tamaños de la carga útil de los paquetes mostrados en la Figura del análisis con *Wireshark* 5.1, se corresponden con los almacenados posteriormente en su respectiva posición de secuencia.

## 5.2. Simulación de clasificación

A continuación se realizará una simulación de clasificación en un entorno controlado, verificando que el sistema de clasificación es eficaz y que el sistema de entrenamiento funciona correctamente.

En primer lugar, se necesita el modelado de los diferentes aplicaciones. Como se explico en el Apartado 4.6, TIE cuenta con la posibilidad de crear sus propias plantillas de flujos etiquetados, generadas por los propios clasificadores. Sin embargo, los clasificadores actualmente disponibles en TIE, o están en fase de desarrollo o no ofrecen una fiabilidad adecuada.

Por lo tanto, para esta simulación se necesita un vía alternativa para obtener estos flujos preetiquetados. Por ello se usará un archivo en formato *pcap* de 1.5 Gbytes de tamaño, capturado por el grupo de investigación del tutor en 2010. Archivo del cual se tienen sus diferentes flujos pre-clasificados.

Este fichero de tráfico contiene en torno a 126.000 sesiones, de las cuales 21.000 son de aplicaciones reconocidas. El desglose de que tipo de aplicación generó estas sesiones se detalla en la Tabla 5.3.

Las más del 100.000 sesiones desconocidas son identificadas por TIE con 0 para el identificador *app\_id* y 0 para *sub\_id*, por lo que durante la fase de entrenamiento generarán sus propios modelos.

Este archivo de tráfico contaba con un fichero en el que cada uno de los distintos flujos almacenados venían etiquetados. Sin embargo, el formato de este fichero, véase la Figura C.1, es diferente al que TIE necesita para la preclasificación de los flujos, similar al que se muestra en la Figura 5.2.



Número de sesiones por aplicación	
Aplicación	Sesiones
HTTP	18.000
SSL	435
Windows messenger	54
Bittorrent	2.000
Netbios	12
NTP	19

Tabla 5.3: Sesiones presentes en el archivo de tráfico externo

Consecuentemente, se tuvo que realizar un procesado de este fichero, programando para ello un conjunto de funciones en código C y en lenguaje *batch*, que se detallan en el Apéndice C.

Una vez que se tiene el archivo de pre-clasificación o *ground truth file*, que se denomina *param\_db\_TIE\_ground\_file.txt* para este tráfico, con cada uno de los flujos etiquetados, se procede con el entrenamiento del clasificador. Como se expuso en el Apartado 4.6.3, existen diferentes variables globales que afectan a la ejecución del clasificador, que pueden modificarse desde un archivo externo al programa *MSBC\_conf.txt*. La utilidad de cada una de estas variables se detalla en el Apéndice B, donde además se indican los valores por defecto de las mismas.

En esta simulación se aplicarán los valores por defecto. Algunos de los más importantes para la comprobación de la correcta operación del sistema son los mostrados en la Tabla 5.4.

Para llevar a cabo el entrenamiento se ejecuta la siguiente orden desde el terminal, en el directorio de instalación de TIE:

```
./TIE -m o -r /traffic/Trafico1.pcap -a MSBC_1.1
-l 20 -u 512 -U 5 -e param_db_TIE_ground_file.txt
```

Valores de las variables de simulación	
Utilidad de la variable	Valor
Tamaño de escalado de mensajes (Variable B en eq. 3.2)	500
Máximo número de gaussianas permitido por modelada de un espacio muestral	15
Modificador dinámico del desplazamiento de los <i>centroides</i> en el algoritmo de bisección	0.1
Mínimo error que ha de recuperarse en una iteración para que tanto el bucle de bisección como el de K-medias clásico no acabe	0.0000002

Tabla 5.4: Principales variables del entorno de simulación

Argumentos empleados en el entrenamiento de MSBC	
Argumento	Utilidad
<b>-m o</b>	Modo de ejecución <i>offline</i>
<b>-r /traffic/Trafico1.pcap</b>	Archivo <i>Trafico1.pcap</i> como entrada de paquetes del programa
<b>-a MSBC_1.1</b>	Clasificador MSBC_1.1 usado en la ejecución
<b>-e param_db_TIE_ground_file.txt</b>	Archivo de entrada con los flujos pre-clasificados ( <i>ground truth</i> )
<b>-l 20</b>	Modo de entrenamiento
<b>-u 512</b>	Indica el mínimo tamaño de la carga útil de un paquete necesario para que dicho paquete no se considere el último de un mensaje
<b>-U 5</b>	Número de mensajes en secuencia necesarios por cada sesión
<b>-E ground_file_test.txt</b>	Fichero donde se almacena la clasificación (flujos ordenados). Esto permite reordenar la lista de flujos etiquetadas respetando el orden que sigue TIE

Tabla 5.5: Argumentos usados para el entrenamiento del *plugin* MSBC\_1.1

**-E ground\_file\_test.txt**

El objetivo de cada uno de los diferentes argumentos especificados en la orden anterior se indica en la Tabla 5.5.

Se puede comprobar (véase la Figura 5.4) durante la ejecución que los parámetros usados son los indicados por defecto, que eran los deseados.

El entrenamiento generará un fichero *own\_MSBC\_signature\_file*, que contendrá los modelos generado para cada una de las aplicaciones. Posteriormente, se realiza la clasificación tomando como referencia los modelos estimados durante el entrenamiento, para ello se ejecuta esta orden en la línea de comandos:

```
./TIE -m o -r traffic/Trafico1.pcap -a MSBC_1.1 -u 512 -U 5
```

En estas orden en la que se observa que los comandos son similares, pero sin hacer referencia a los ficheros donde se registra la preclasificación.

El resultado de la clasificación de una ejecución de TIE, esto es, cada uno de los flujos analizados con una etiqueta que le atribuye la aplicación que lo generó, se almacena en el fichero *class.tie*, si no se indica otro con el comando argumento -E. Este fichero de salida se puede comparar con el fichero de entrenamiento, en el que los flujos estaban previamente clasificados. Para

```

clusters required: 15
Normalized size: 500
Min probability given for a cluster without gaussian: 0.000000010
Euclidean_mean: Yes
Euclidean_shift: Yes
Kmeans model: 2,          one|square
Kmeans_fixed_shift: 0.1000
Kmeans_dynamic_step_modifier: 0.1000
Error_threshold: 0.0000002000
Min_gaussian_variance: 0.0000001000
Print_map_file: Yes
Signature_file_name: own
Number of protocolo to train: 408
Kmeans calculate probability : 2,          max|total

```

Figura 5.4: Captura parámetros por defecto en una ejecución del clasificador

ello, TIE dispone de una serie de scripts útiles para este fin, en este caso se usará el script *TIE\_stats*. La orden para usar este *script* y referenciar los ficheros es la siguiente (suponiendo que están los archivos en el directorio del script ejecutado):

```
./TIE_stats -p ground_file_test.txt.tie class.tie
```

Orden en la que con *"-P ground\_file\_test.txt.tie"* se hace referencia al archivo con el que se compararán los resultados, tomando como verídicos los almacenados en dicho fichero. Este archivo es el generado durante la fase de entrenamiento, ya que el otro archivo usado en esta fase tenía los flujos en un orden distinto al que TIE usa en sus plantillas (este es el archivo generado durante el procesamiento del tráfico externo, Apéndice C).

Este *script* genera una serie de estadísticas acerca de la clasificación realizada. En primer lugar, nos informa del número de sesiones, de paquetes y de bytes que se han procesado en la ejecución, véase la Figura 5.5.

Uno de los datos más importantes es el porcentaje de sesiones identificadas (*Accuracy* en la Figura 5.6), que se puede observar en la esquina superior izquierda de dicha imagen. Este valor indica el número de sesiones identificadas con éxito, esto es, con respecto al fichero de clasificación tomado como referencia, cuántas sesiones han sido etiquetadas con los mismos identificadores. En esta estadística no intervienen las sesiones con etiqueta de aplicaciones desconocidas (*unknown*).

En esta ejecución ha sido del 87.96 %, que no es una cifra muy elevada, pero que sí que confirma la funcionalidad del algoritmo implementado para llevar a cabo la identificación de tráfico.

Seguidamente, ofrece una tabla en la que se contabilizan las sesiones de

```

General statistics
-----
Sessions: 126 K (TCP: 57 K, UDP: 68 K, Other: 0)
Packets: 631 K (dstream: 177 K, upstream: 453 K)
Bytes:      87 M (dstream: 54 M, upstream: 32 M)

```

Figura 5.5: Estadísticas del tráfico procesado

```

Per app statistics (from /home/jkant/Documents/TFG/tie1.2/bin/output/ground_file_test.txt.tie)
-----
Known apps: 7
Accuracy: 87.96 %
-----

```

Label	ID	Matched sessions	Sessions Known	Sessions Found	Packets Known	Packets Found	Bytes Known	Bytes Found
HTTP	1	90.5 %	18 K	16 K	230 K	229 K	66 M	67 M
SSL	121	93.3 %	435	406	6 K	6 K	2 M	2 M
WINDOWS_MESSENGER	152	81.5 %	54	44	783	660	127 K	91 K
BITTORRENT	132	69.0 %	2 K	25 K	33 K	83 K	1 M	9 M
NETBIOS	28	48.5 %	132	90	530	796	52 K	80 K
NTP	291	0.0 %	12	0	19	0	912	0
	300	15.8 %	19	3	266	49	129 K	33 K
UNKNOWN	0	77.7 %	105 K	83 K	358 K	309 K	16 M	9 M

```

-----
Confusion Matrix (apps) (by session):
-----

```

	UNKNOWN	HTTP	SSL	WINDOWS_	BITTORRE	NETBIOS	NTP
UNKNOWN	[77.7]	0.2	-	-	22.2	-	-
HTTP	5.5	[90.5]	-	-	4.0	-	-
SSL	3.9	2.5	[93.3]	-	0.2	-	-
WINDOWS_	3.7	11.1	-	[81.5]	3.7	-	-
BITTORRE	17.5	12.4	-	-	[69.0]	1.1	-
NETBIOS	36.4	7.6	-	-	7.6	[48.5]	-
NTP	100.0	-	-	-	-	-	-
	5.3	78.9	-	-	-	-	[15.8]

Figura 5.6: Matriz de aciertos para una simulación con los valores por defecto de sus variables

cada tipo de aplicación encontradas en el archivo de referencia (*Known*), y las que se han encontrado en el archivo de clasificación que se pretende evaluar (*Found*). Además, ofrece el porcentaje de sesiones que han sido identificadas para cada aplicación.

Tras esta tabla, genera una matriz denominada *Confusion Matrix*, útil para evaluar las fugas entre las distintas aplicaciones, es decir, que porcentaje de sesiones generadas por una aplicación que han sido clasificadas como generadas por otra.

Si observamos esta matriz detenidamente se tiene que:

- El mayor porcentaje de identificación se tiene para conexiones SSL, alrededor del 93.3 %.
- Para NTP (Network Time Protocol) no se ha reconocido ninguna sesión. Esto es debido a que para entrenar una aplicación, debe de haber un flujo cuyo intercambio de paquetes genere al menos 5 mensajes (como se define en la orden con el comando *-U*), sin embargo, para esta aplicación no se llega a este número de mensajes. Su archivo de muestras temporal, *291\_0\_app.txt*, se encuentra vacío.
- Para HTTP y Windows Messenger se obtiene un respetable porcentaje de sesiones identificadas.
- Hay un último protocolo para el que no aparece el nombre, que es DDL (Direct Download Link), que tiene unos pésimos resultados, en torno al 15.8 %.

- Para la aplicación P2P bittorrent se tiene un porcentaje del 70 %. La principal fuente de error para esta aplicación es la aplicación nombrada como *unknown*, que agrupa a aquellas sesiones de las que no se conocía su procedencia previamente. Consecuentemente, es posible que este conjunto de sesiones mal etiquetadas contenga parte de P2P.

Esta simulación no proporciona unos datos demasiados fiables sobre el método implementado, ya que el tráfico usado para el entrenamiento es el mismo que el se ha usado para verificar el funcionamiento del clasificador. En un escenario real, el tráfico a clasificar será diferente al usado durante la fase de entrenamiento.

No obstante, si que demuestra que los modelos generados durante el entrenamiento, usados en la posterior clasificación funcionan correctamente, así como que la funcionalidad de clasificación también funciona de manera eficaz, ya que los resultados obtenidos son coherentes.

### 5.3. Simulaciones variando variables que regulan el sistema

Una vez comprobado que el sistema funciona correctamente, ya que es capaz de dar unos resultados aceptables y coherentes, se procederá a realizar diferentes simulaciones, con el objetivo de observar cómo afecta al sistema el uso de uno u otro valor en cada una de las diferentes variables que regulan el método de clasificación, así como aquellas implicadas en el algoritmo de entrenamiento.

Tal y como ya se ha mencionado, este conjunto de variables están agrupadas en una estructura a la que se puede acceder desde cualquier punto del clasificador, ya que ha sido definida como global. La composición de esta estructura se detalla en el Apéndice B. En este apartado se pretende variar el valor de algunas de estas variables con respecto a su valor dado por defecto, comprobando los resultados obtenidos para esa nueva configuración.

#### 5.3.1. Número de gaussianas permitido y número de mensajes iniciales necesarios por sesión

En este apartado se realizan una serie de pruebas para evaluar la mejor configuración para dos variables: el máximo número de gaussianas permitidas para modelar cada conjunto de muestras, valor especificado por la variable *MSBCv.clusters\_required* y, por otro lado, el número de mensajes iniciales en secuencia necesario por cada flujo para llevar a cabo la clasificación.

Este segundo valor se indica desde la línea de comandos (véase la Tabla 4.2), y no desde el fichero de configuración *MSBC\_conf.txt*, ya que el bloque extractor de parámetros necesita conocer el número de mensajes necesarios por sesión, previamente a la clasificación o al entrenamiento. Otro valor

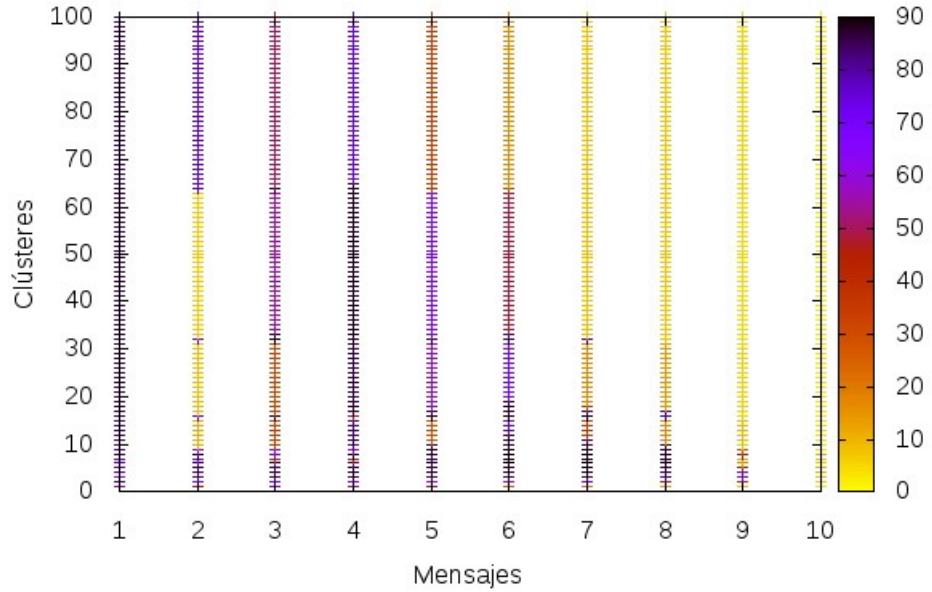


Figura 5.7: Porcentaje de identificaciones correctas en función del número de mensajes analizados por conexión y del máximo número de gaussianas por modelo

pasado por la línea de comandos, es el tamaño mínimo de la carga útil de un paquete necesario para que no se considere dicho paquete el último del mensaje evaluado, este valor estará fijado a 512 bytes.

El resto de las variables, presentes en la estructura *MSBCv* (véase Apéndice B, contendrán su valor por defecto, dado en la Tabla B.1.

Se generarán una serie de códigos que se encarguen de hacer variar estas dos variables de forma conjunta. Por otro lado, se creará otro código, en lenguaje GAWK, para extraer el porcentaje de sesiones clasificadas, valor de *Accuracy* en la Figura 5.6), correctamente a partir del fichero en el que se almacena la evaluación de la clasificación generada por el script de comparación, *TIE\_stats*. Una vez que se obtienen estos valores se representan, obteniéndose la gráfica de la Figura 5.7.

En dicha gráfica generada se tiene una escala de color (*amarillo-rojo-azul-negro*) que indica el porcentaje de sesiones identificadas para cada una de las posibles combinaciones, siendo el color negro indicativo de que se han identificado el 90 % de las sesiones, mientras que el amarillo indica que no se ha identificado ninguna correctamente.

De esa gráfica se pueden extraer una serie de observaciones que son:

- Para un modelo basado solo en el primer mensaje de la comunicación se obtienen, a simple vista, unos resultados de identificación aceptables. Incluso, se aprecia cómo los resultados son mejores para un mensaje que para 2 o para 3 mensajes. En este sentido, también se observa como, para valores del número de mensajes elevado, a partir de 7, el porcentaje de identificación decae.

En principio, esto no es lo que se esperaba de esta simulación, ya que a un mayor número de mensajes mejor será el modelo que caracteriza cada aplicación y, por lo tanto, una herramienta más apropiada para la identificación de los flujos generados por dichas aplicaciones. Sin embargo, el modelo de entrenamiento implementado solo utiliza las muestras de tamaño de mensaje de aquellos flujos que llegan al número necesario de mensajes iniciales, esto es, que a mayor número de mensajes necesarios, más difícil será que un flujo llegue a esa cifra.

Por lo tanto, aunque durante la clasificación aquellos flujos que tengan un número de mensajes menor al solicitado sí que podrán ser clasificados por el procedimiento de reclasificación explicado en el Apartado 4.5.1, la clasificación realizada será incorrecta, ya que no se tienen modelos para la aplicación que generó esos flujos, debido al elevado número de mensajes necesarios.

- Por otro lado, se observa que conforme se aumenta el número de *clusters*, de forma global, se disminuye la precisión. Esto puede estar relacionado con el hecho de que se definen demasiadas distribuciones y algunas de ellas no son representativas de la aplicación.

Además, el conjunto de flujos etiquetados como *unknown* es probable que también influya en este resultado, ya que al dejar que el límite de gaussianas aumente por modelo, se tendrá que muchas aplicaciones no usarán tantas gaussianas porque no tienen tantos métodos definidos, mientras que para los flujos *unknown* se definirán el máximo de gaussianas, provocando que muchos flujos se ajusten a estas nuevas gaussianas generando más errores.

Por lo tanto, se evidencia en la gráfica anterior cómo una buena elección del número de mensajes, así como del número de *clusters* es decisivo con en el resultado final.

Un aspecto interesante antes observado, es que dado un solo mensaje para analizar se obtienen buenos resultados. Por ello, se decide hacer otra simulación con el objetivo de profundizar en las causas de este efecto. En esta nueva simulación se tomarán los valores por defecto de la Tabla B.1 y un solo el mensaje inicial de cada flujo.

Known apps: 7  
Accuracy: 85.79 %

Label	ID	Matched sessions	Sessions		Packets		Bytes	
			Known	Found	Known	Found	Known	Found
HTTP	1	88.5 %	18 K	48 K	230 K	367 K	66 M	69 M
SSL	121	92.6 %	435	1 K	6 K	15 K	2 M	2 M
WINDOWS_MESSENGER	152	70.4 %	54	457	783	5 K	127 K	1 M
BITTORRENT	132	61.8 %	2 K	39 K	33 K	117 K	1 M	13 M
NETBIOS	28	87.1 %	132	117	530	463	52 K	45 K
NTP	291	100.0 %	12	34	19	43	912	2 K
	300	84.2 %	19	101	266	1 K	129 K	434 K
UNKNOWN	0	33.4 %	105 K	36 K	358 K	123 K	16 M	0

Confusion Matrix (apps) (by session):

	UNKNOWN	HTTP	SSL	WINDOWS_	BITTORRE	NETBIOS	NTP
UNKNOWN	[33.4]	30.7	0.1	0.2	35.5	-	-
HTTP	4.6	[88.5]	3.7	0.7	2.1	-	0.5
SSL	1.1	1.1	[92.6]	-	5.1	-	-
WINDOWS_	-	16.7	3.7	[70.4]	9.3	-	-
BITTORRE	16.1	15.6	4.0	2.5	[61.8]	-	-
NETBIOS	-	-	0.8	-	12.1	[87.1]	-
NTP	-	-	-	-	-	-	[100.0]
	5.3	10.5	-	-	-	-	[84.2]

Figura 5.8: Resultados de una simulación para un solo mensaje analizado por comunicación

Una vez realizada la simulación, el resultado obtenido, desglosado en la "matriz de confusión" para las diferentes aplicaciones, se muestra en la Figura 5.8.

Si comparamos los resultados con los de la simulación realizada para un número de 5 mensajes iniciales necesarios (Figura reffss2), se tiene que para las aplicaciones para las que se obtenían unos buenos resultados, ha disminuido el porcentaje de flujos bien clasificados, mientras que para las demás ha aumentado considerablemente. Para poder comparar convenientemente estos resultados se construye la Tabla 5.6.

En dicha tabla se observa que:

- Para un único mensaje analizado se obtiene un elevado número de sesiones para las aplicaciones NetBIOS, DDL y NTP.
- Para la "mezcla de aplicaciones" que no estaban identificadas, *Unknown*, se tiene que disminuyen drásticamente las identificaciones correctas.
- Disminuye levemente el porcentaje de acierto para HTTP, SSL y Windows\_messenger. Es de esperar que al tener menos mensajes que analizar estas aplicaciones queden peor caracterizadas. Sin embargo, es muy pequeña la bajada de aciertos para lo que se podría esperar.

No obstante, si se observan detenidamente los resultados para ambas simulaciones, se tiene que para la simulación de un solo mensaje la precisión disminuye considerablemente con respecto a la anterior, entendiéndose precisión como el número atribuciones correctas entre el número de atribuciones



Aplicación	Estadísticas de atribuciones					
	5 mensajes			1 mensaje		
	Iden. (%)	Asig.	Espe.	Iden. (%)	Asig.	Espe.
<b>Unknown</b>	77.7	83k	105k	33.4	36k	105k
<b>HTTP</b>	90.5	16k	18k	88.5	48k	18k
<b>SSL</b>	93.3	406	435	92.6	1k	435
<b>MSN</b>	81.5	44	54	70.4	457	54
<b>Bittorrent</b>	69.0	25k	2k	61.8	39k	2k
<b>NetBios</b>	48.5	90	132	87.1	117	132
<b>NTP</b>	0	0	12	100	34	12
<b>DDL</b>	15.8	3	19	84.2	101	19

**Iden.** Atribuciones identificadas

**Asig.** Atribuciones asignadas

**Espe.** Atribuciones esperadas

Tabla 5.6: Estadísticas de atribuciones para 1 y 5 mensajes

totales<sup>1</sup>, que son aquellas sesiones que se han atribuido a esa aplicación aún sin saber si dicha atribución es correcta o no.

El número de atribuciones correctas, vendrá dado por el número de atribuciones esperadas, por el porcentaje de aciertos (divido entre 100 para normalizar). Por lo tanto la expresión para calcular la precisión quedaría:

$$Precision = \frac{Atribuciones_{esperadas} * \frac{\%Iden.}{100}}{Atribuciones_{Asignadas}}$$

Aplicando esta relación se obtiene la Tabla 5.7. Donde se observa de forma evidente como para la simulación con 5 mensajes hay mucha mas precisión que para la de un solo mensaje. Únicamente para NetBIOS se consiguen unas tasas tanto de precisión como de identificación elevadas.

Resumiendo, para una clasificación basada en el estudio de solo el primer mensaje de cada flujo se obtienen unos niveles respetables de identificación, debido a que se hace una clasificación imprecisa, atribuyendo a demasiadas sesiones aplicaciones que no las generaron.

Sin embargo, el aumento de identificaciones correctas realizadas para un solo mensaje puede estar relacionado con que algunos flujos no llegan a la cantidad necesaria de mensajes para ser analizados, provocando que no se

<sup>1</sup>Normalmente cuando se habla de precisión se hace referencia al número de identificaciones correctas (verdaderos positivos (*VP*)) entre el número de identificaciones totales (falsos positivos (*FP*) y verdaderos positivos (*VP*)).  $Precisión = \frac{VP}{VP+FP}$ .

Aquí se han adaptado los términos a los facilitados por el *script* de comparación, *TIE-stats*

Estadísticas de atribuciones				
Aplicación	5 mensajes		1 mensaje	
	Iden. (%)	Prec. (%)	Iden. (%)	Prec. (%)
<b>Unknown</b>	77.7	98.2	33.4	97.4
<b>HTTP</b>	90.5	100	88.5	33.1
<b>SSL</b>	93.3	99.5	92.6	40.2
<b>MSN</b>	81.5	100	70.4	8.3
<b>Bittorrent</b>	69.0	5.6	61.8	2.2
<b>NetBios</b>	48.5	71.13	87.1	98.2
<b>NTP</b>	0	-	100	35.2
<b>DDL</b>	15.8	100	84.2	15.8

**Iden.** Atribuciones identificadas

**Prec.** Precisión en las atribuciones

Tabla 5.7: Estadísticas de identificación y precisión para cada aplicación para 1 y 4 mensajes

genere ningún modelo para la aplicación que genera esos flujos como para NetBIOS, generando pésimos resultados para una análisis de 5 mensajes.

Aunque esta simulación se ve afectada por algunos factores, como son el poco número de flujos utilizados, así como la diferencia de muestras para la caracterización de cada aplicación, sí que resulta evidente qué se deberá re-estructurar el procedimiento del entrenamiento, habilitándose la recogida de muestras para flujos de cualquier tamaño de mensaje, y por lo tanto, el posterior entrenamiento de estas muestras, generándose así modelos para esos flujos de pocos mensajes.

### 5.3.2. Tamaño mínimo de carga útil

En este experimento se realizarán una serie de simulaciones variando el valor del tamaño mínimo necesario de carga útil de un paquete para que este paquete no sea considerado el último del mensaje evaluado.

Cabe recordar que esta variable no se indicaba desde la estructura de variables globales, sino desde la orden de ejecución de TIE, utilizando el argumento habilitado para ello (Figura 4.2). El resto de variables tendrán su valor indicado por defecto (véase la Tabla B.1, mientras que el número de mensajes a analizar, que también se indicaba por línea de comandos, será fijado a su valor habitual de 5 mensajes.

En la Figura 5.9 se observa que el uso de un tamaño u otro es prácticamente irrelevante para el parámetro observado. Se obtiene un valor en torno al 88.5% de sesiones identificadas correctamente para todos los valores de este tamaño.

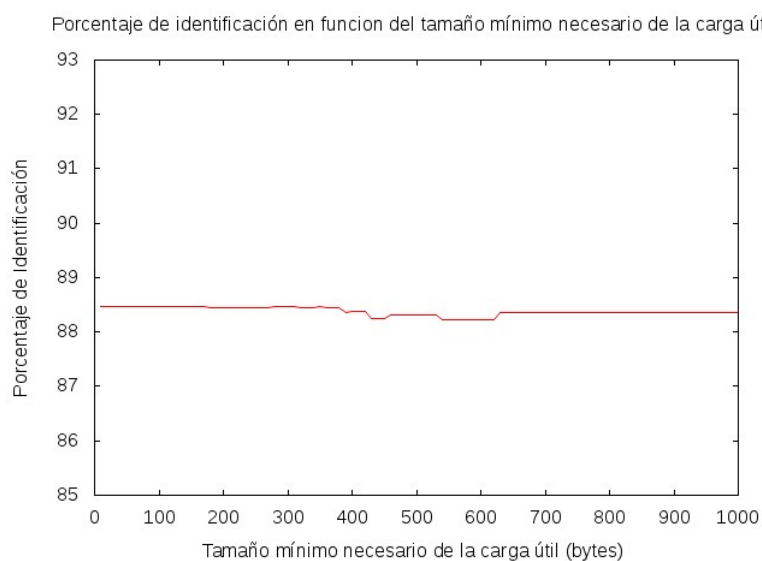


Figura 5.9: Porcentaje de sesiones correctamente identificadas en función del mínimo tamaño de carga útil del paquete para que no acabe el mensaje actual

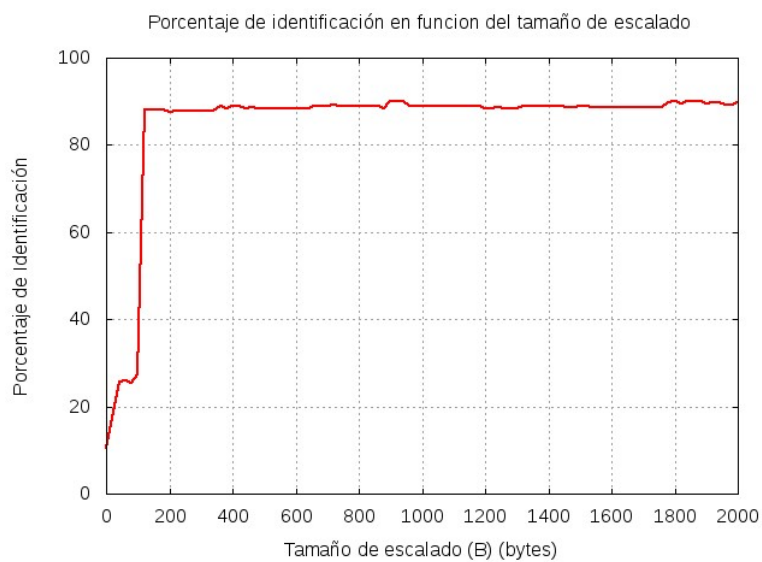


Figura 5.10: Porcentaje de sesiones correctamente identificadas en función del mínimo tamaño de carga útil del paquete para que no se de por evaluado el mensaje

### 5.3.3. Valor utilizado para escalar el tamaño de cada mensaje

En estas simulaciones se variará el valor de la variable global *MSBCv.normalized\_size*, la cual definía el valor que utilizado para escalar el tamaño de los diferentes mensajes procesados. Este proceso de escalado quedó explicado en el apartado 3.2.2, siendo dependiente del valor, B, de esta variable.

$$s'_i = \frac{s_i}{(B + |s_i|)}$$

El resto de variables globales definidas tendrán su valor por defecto (Tabla B.1), mientras que las otras dos que se indican por la línea de comandos estarán también en su valor habitual de 5 mensajes para analizar y 512 bytes como el mínimo tamaño de carga útil necesario para que el mensaje actual no acabe.

De nuevo, si se observa la Figura 5.10 se tiene una variable que no influye demasiado en el comportamiento del sistema si no se eligen unos valores excesivamente pequeños. En los resultados se obtiene un máximo del 90.80 % de las sesiones identificadas para un valor de escalado de 1800 Bytes.

## Capítulo 6

# Conclusiones y futuro trabajo

En este trabajo se ha implementado un clasificador que incorpora una nueva metodología de identificación de tráfico a las ya existentes en TIE. Algunos de los aspectos a destacar, así como algunas de las contribuciones realizadas durante el desarrollo del trabajo son las siguientes:

- Se ha implementado un módulo de clasificación basado en el tamaño de los mensajes. Adicionalmente, se ha desarrollado para este módulo la posibilidad de clasificar cada sesión cuando se recibe el primer mensaje, re-clasificándose si llega otro mensaje hasta que se llega al número de mensajes requerido. Esto disminuye de forma significativa el número de sesiones que no se pueden clasificar por falta de mensajes analizados.
- Se ha diseñado e implementado, modificándose la estructura de TIE, una serie de módulos que permiten la extracción del tamaño de los mensajes de cada sesión o flujo.
- Se ha diseñado e implementado un módulo de entrenamiento para la metodología de clasificación de tráfico presentada en este trabajo. Este módulo implementa un algoritmo de agrupamiento basado en K-medias, modificado con objeto de adaptarlo al propósito de generar los modelos de cada aplicación.

Este módulo de entrenamiento ha sido habilitado para generar el modelo de todas las aplicaciones definidas para la versión 1.2 de TIE.

Por otra parte, el algoritmo K-medias implementado depende de una serie de variables que permiten regular y ajustar su funcionamiento.

- El clasificador consta de un archivo de configuración externo que permite modificar y regular las diferentes variables del clasificador, evitando así tener que compilar de nuevo el sistema si solo se quiere

variar una de esas variables, a la vez que se permite evaluar diferentes configuraciones de una forma más sencilla.

- Adicionalmente, se han generado una serie de programas necesarios para adaptar al formato de entrada de TIE para los archivos preclasificados, *ground\_file*, los ficheros de clasificación generados por otra aplicación externa a TIE.

Con respecto a la metodología implementada, los resultados obtenidos son aceptables para algunas aplicaciones, como SSL o HTTP, pero para algunas no resultan suficientes. Aunque la principal característica positiva del método es la rapidez con la que se identifica el tráfico, es necesario ajustar y analizar el funcionamiento del método para mejorar las tasas de identificaciones correctas.

## 6.1. Trabajo futuro

Durante el desarrollo de este trabajo, en la implementación tanto del módulo clasificador como el módulo entrenador, se ha hecho uso del tamaño real de los mensajes, por lo que cuando se quería estimar la diferencia entre dos mensajes se tenía que transformar el tamaño de dichos mensajes al espacio escalado. Por ello, una posible modificación sería escalar el tamaño de cada mensaje previamente a ser almacenado, ahorrando trabajo y tiempo para el resto del proceso.

El módulo de entrenamiento implementado recoge muestras de tamaño de mensaje solo si una sesión llega al número de mensajes necesario para el entrenamiento. Sin embargo, como se vio en la simulación del apartado 5.3.1, algunas aplicaciones no producen flujos que lleguen a ese número de mensajes, por lo tanto imposibilita que se recojan muestras de esas aplicaciones, quedándose sin modelar.

Consecuentemente, una mejora que afectaría a la tasa de identificaciones correctas, sería habilitar el almacenamiento de muestras para cualquier sesión. Esto posibilitaría que durante la fase de clasificación se tuvieran diferentes modelos según el número de mensajes de la sesión que se pretende clasificar.

Como ya se ha mencionado, en este trabajo se han programado una serie de herramientas que permiten adaptar al formato de lectura de archivos de preclasificación de TIE, una serie de ficheros con flujos etiquetados generados por otra aplicación. De este tráfico ya clasificado, solo se ha usado 1.5 GB, siendo el total del tráfico disponible cercano a 1 TB. Consecuentemente, haciendo uso de esa serie de programas desarrollados se tendría una gran biblioteca de tráfico, dispuesta para ser utilizada en el proceso de entrenamiento, generando así unos modelos de las aplicaciones más realistas.

Por otra parte, durante el desarrollo de este trabajo se ha tenido que modificar el núcleo de TIE, ya que se necesitaban extraer una serie de parámetros que TIE no nos facilitaba. Consecuentemente, sería necesario revisar esta plataforma para que habilitara que cada clasificador pudiera acceder al extractor de parámetros, así como a la estructura de datos almacenada para cada sesión con el objetivo de almacenar los parámetros que cada clasificador necesite.





# Apéndice



## Apéndice A

# Extracción de los modelos externos

Al comenzar el proyecto, se nos proporcionó un conjunto de ficheros que contenían el modelado basado en distribuciones de diferentes aplicaciones. Este modelado seguía un esquema diferente al formato que se definió para la lectura de los modelos de las distintas aplicaciones, que quedó expuesto en el Apartado 4.4.2.

Consecuentemente, se generaron una serie de funciones para adaptar estos ficheros al formato requerido. En este apéndice se explicará el código implementado para este fin y, posteriormente, se expondrá que estos modelos no han proporcionado buenos resultados de clasificación, analizándose las razones a lo que esto puede deberse.

### A.1. Análisis del formato del modelado

A diferencia del formato diseñado para el clasificador, estos modelos estaban repartidos en diferentes ficheros. Cada fichero almacenaba el modelo para una dirección de mensaje de una posición de secuencia de una sola aplicación.

El nombre de estos ficheros es: *n\_l\_T\_d.csv*, donde la *n* indica la aplicación, *l* la posición en la secuencia de mensaje, *T* el protocolo de la capa de transporte (TCP/UDP) y *d* la dirección, *downstream* o *upstream*. Para conocer a qué aplicación se refiere el índice *n*, existe un fichero denominado *protos.txt*. Sin embargo, no todas las aplicaciones enumeradas en dicho fichero tienen modelo. En la Tabla A.1 se indica la atribución de cada índice con las diferentes aplicaciones que sí que tienen modelado.

Por otra parte, cada uno de estos ficheros contiene los datos distribuidos de la forma que se muestra en la Figura A.1. Donde cada gaussiana estimada de ese modelo *clúster* contiene los siguientes parámetros:

- Un número identificativo de cluster, en orden, 1,2,3....k.

Atribución de identidad	
ID	Aplicación
1	Bittorrent
2	iMesh
8	Messenger
16	SSL
17	HTTP
21	Gnutella
27	Oscar
32	Pop3
34	FTP
36	SNMP

Tabla A.1: Atribución del número de identidad para los clústeres externos

- Número de muestras agrupadas por esa gaussiana
- *Centroide* de la gaussiana.
- Valor del peso relativo.
- El último valor es la varianza asociada a la gaussiana
- Finalmente, incluye una lista con todas las muestras agrupadas por esa gaussiana.

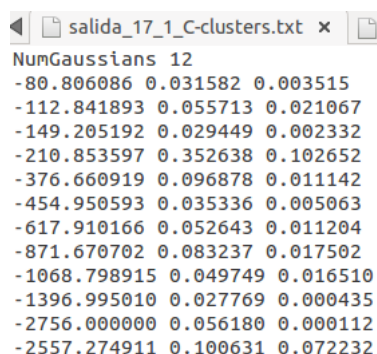
En principio, se tienen todos los elementos necesarios para el formato definido del modelo para el clasificador, ya que solo necesita del *centroide*, de la varianza y del peso relativo de cada gaussiana dentro del modelo total (gausiana multiplico).

```

17_1_C-clusters.csv x TheFormat.txt x
1,3,-80.806086,0.031582,0.003515
,,,,,-80.000000,-85.000000,-79.000000,
2,3,-112.841893,0.055713,0.021067
,,,,,-102.000000,-115.000000,-102.000000,
3,3,-149.205192,0.029449,0.002332
,,,,,-152.000000,-149.000000,-147.000000,
4,22,-210.853597,0.352638,0.102652
,,,,,-202.000000,-219.000000,-306.000000,-2
5,9,-376.660919,0.096878,0.011142
,,,,,-355.000000,-397.000000,-356.000000,-3

```

Figura A.1: Extracto de un fichero de almacenamiento de modelos generados por la aplicación externa a TIE



```
NumGaussians 12
-80.806086 0.031582 0.003515
-112.841893 0.055713 0.021067
-149.205192 0.029449 0.002332
-210.853597 0.352638 0.102652
-376.660919 0.096878 0.011142
-454.950593 0.035336 0.005063
-617.910166 0.052643 0.011204
-871.670702 0.083237 0.017502
-1068.798915 0.049749 0.016510
-1396.995010 0.027769 0.000435
-2756.000000 0.056180 0.000112
-2557.274911 0.100631 0.072232
```

Figura A.2: Salida generada al aplicar el código de comandobeta.awk sobre 17\_1\_C-cluster.csv

## A.2. Procesado de cada uno de los ficheros

Como se ha mencionado anteriormente, de cada fichero deberemos extraer una serie de elementos y, posteriormente, imprimir el valor de esto en el futuro fichero que guardará los diferentes modelados, que en este caso se llamará *foreign\_signature\_file.txt*. Para la extracción de estos elementos se genera un código en lenguaje AWK.

Este código se encarga de, dado un fichero como el de la Figura A.1, tomar para cada gaussiana los valores necesarios para el modelo del método implementado, esto es, el *centroide*, la varianza y el peso relativo. Este código generado, en un fichero llamado *comandosbeta.awk*, se ejecuta con la orden:

```
gawk -f comandosbeta.awk 17_1_C_-clusters.csv >
salida_17_1_C-clusters.txt
```

En el archivo generado, Figura A.2, se indica, en primer lugar, el número de gaussianas presentes en el modelo y, seguidamente, se dedica cada columna para cada uno de los diferentes parámetros necesarios por gaussiana, *centroide*, peso relativo y varianza.

El formato del fichero anterior es similar al definido para el clasificador para cada uno de los modelados de una aplicación, para una posición de mensaje en la secuencia y de una dirección. Por lo tanto, resta recorrer el conjunto de ficheros, generando el archivo *foreign\_signature\_file.txt*.

## A.3. Creación del fichero *foreign\_signature\_file.txt*

Para construir el fichero *foreign\_signature\_file.txt* se recorrerá cada uno de los distintos ficheros que contienen los modelos. Para ello, se crearan una serie de bucles con el objetivo de direccionar estos ficheros, aplicándoles el código de *comandosbeta.awk* y almacenando el resultado, respetando el orden definido para el fichero que se quiere generar en la sección 4.4.2.

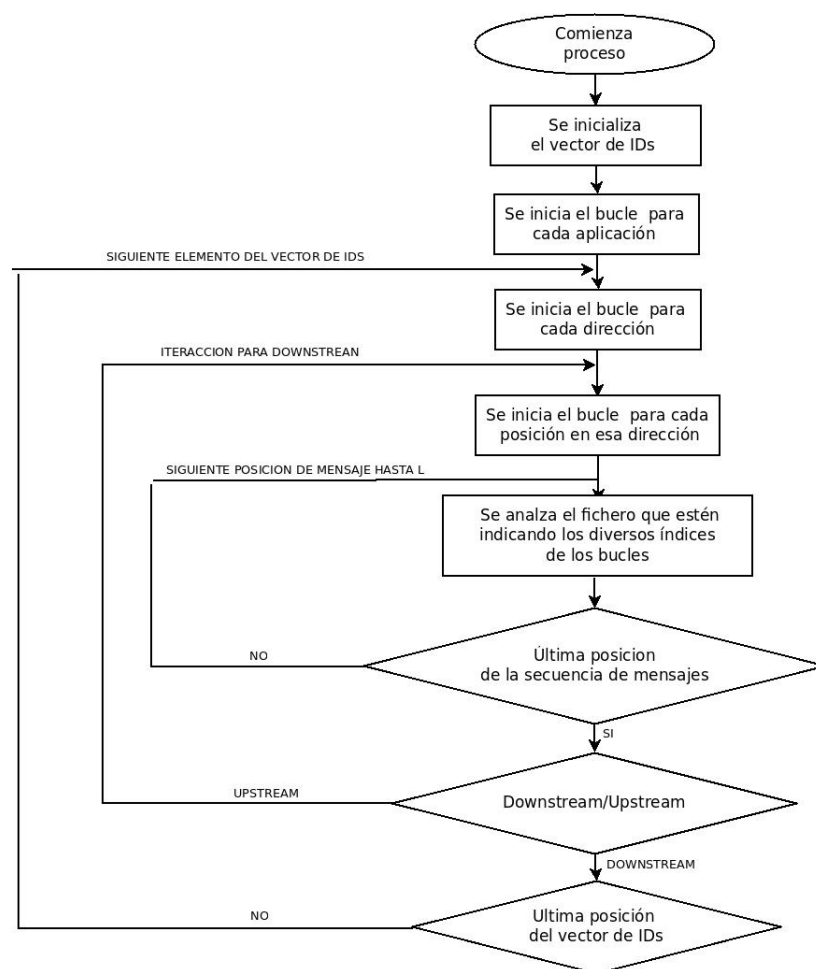


Figura A.3: Estructura de bucles enlazados en el fichero `scriptFormatos.sh`

La estructura de los bucles enlazados es la que se puede observar en la Figura A.3, diseñada en lenguaje *Batch*, siendo el nombre del fichero `scriptFormatos.sh`.

El código antes mencionado se ejecuta con la orden:

```
bash scriptFormatos.sh > foreign_MSBC_signature_file.txt
```

Además del fichero en el que se almacenarán todos los modelos con la orden anterior, `foreign_MSBC_signature_file`, hace falta otro que para asociar la identidad de cada aplicación (Tabla A.1), con sus números de identificación en TIE, `app_id` y `sub_id` de TIE. Este fichero es el que se denominó como `foreign_MSBC_map_file.txt`. Dicho fichero se muestra en la Figura A.4.

```

IdentifiedProtocols 38
#MSBC_ID      App_ID  Sub_ID  Label
0,      331,    0,      MySQL
1,      132,    0,      Bittorrent
2,      130,    0,      iMESH
3,      303,    0,      PANDO
4,      0,      0,      Windowsmedia
5,      0,      0,      unknown
6,      20,    0,      SSH
7,      28,    1,      SMB
8,      152,    0,      MSN
9,      212,    0,      IPP
10,     148,    0,      SIP
11,      1,    12,     MPEG
12,      5,    0,      DNS
13,     57,    0,      NTP
14,     326,    0,      GRE
15,     57,    0,      IRC
16,     121,    0,      SSL
17,      1,    0,      HTTP
18,     179,    0,      RDP
19,     131,    0,      DirectConnect

```

Figura A.4: Archivo *foreign\_MSBC\_map\_file.txt*, habilita la asociación entre la identificación de cada modelo con sus números identificativos de TIE *app\_id* y *sub\_id*

#### A.4. Simulación de clasificación con este modelado

Para comprobar la utilidad del conjunto de modelos extraídos, se va a realizar una simulación de clasificación sobre el tráfico del cual se dispone un fichero en el que están todos los flujos pre-clasificados, el mismo que se utiliza para el entrenamiento.

Se ejecutará la simulación respetando los valores por defecto, pero cambiando el tipo de origen del modelado de *own* a *foreign*, ya que en el directorio de instalación del *plugin* deberán estar tanto los ficheros de *own* como los de *foreign*, posibilitando las dos opciones.

Realizada la simulación, se compararán los resultados con el archivo

```

Confusion Matrix (apps) (by session):
-----
          UNKNOWN      HTTP      SSL WINDOWS_ BITTORRE
UNKNOWN  [33.4]        0.5        0.4    65.5      -
HTTP      4.6      [6.2]        -    88.5      0.7
SSL        1.1      46.2        -    28.3     24.1
WINDOWS_   -      18.5        -    [75.9]      1.9
BITTORRE  16.1      14.8      2.2    66.2     [0.4]
NETBIOS    -       8.3        -    91.7      -
NTP        -       -        -   100.0      -
          5.3      10.5        -    84.2      -

```

Figura A.5: Resultados de una simulación usando el modelado generado por la aplicación externa a TIE

en el que estaban pre-clasificados los flujos, para evaluar la eficacia de la clasificación. Para ello, se usa un *script* habilitado para este fin. El resultado de esta comparación puede apreciarse en la Figura A.5, donde se verifica que la clasificación realizada no ha sido realista, puesto que los resultados son bastante negativos.

Estos malos resultados pueden deberse a estos factores:

- La forma de de fijar la dirección de la comunicación es diferente a la implementada en el extractor de parámetros diseñado. Por ejemplo, para SMTP vienen definidos una serie de modelos en los que para el primer mensaje intercambiado solo hay gaussianas definidas para el sentido *upstream*. Sin embargo, para esta aplicación el primer mensaje con contenido es el *HELO* del servidor, siendo esto detectado como sentido *downstream* por nuestro clasificador, ya que el primer mensaje lo envía el cliente durante el *handshake* de TCP.
- Por otro lado, se observa que en muchos de los modelos de gaussianas multiplico se tiene que la suma de los pesos relativos de cada una de las gaussianas no suma 1, que es el resultado esperado. Por lo tanto, se tiene una incoherencia con respecto al método implementado.

Consecuentemente, para lograr que este conjunto de modelos sea compatible con el clasificador implementado se necesitaría conocer más información acerca de como se han estimado, para así poder adecuar la forma en que son tratados estos modelos.



## Apéndice B

# Archivo de configuración *MSBC\_conf.c*

Este archivo se ha diseñado con objeto de tener acceso a los diferentes parámetros del *plugin* de una forma simple y sin tener que acceder al código en C del programa.

Los parámetros a los que se accede desde este archivo forman parte de una estructura global del clasificador, Figura B.1, disponible desde cualquier punto de este.

A esta estructura se la ha llamado *MSBCv*, por lo que todas las variables contenidas serán denominadas *MSBCv.xxx*, siendo *xxx* el nombre de la variable dentro de la estructura.

### B.1. Propósito de cada variable

En este apartado se van a enumerar las distintas variables englobadas en la estructura antes mencionada, comentando brevemente el propósito de

<b>struct MSBC_variables</b>
normalized_size: uint_16
signature_file: string
calculate_probability: int
no_gaussian_available: float
print_map_file: bool
num_protocols_train: uint_16
clusters_required: int
minimum_error_recoverable: float
euclidean_shift: bool
fixed_shift: float
step_modifier: float
euclidean_mean: bool
min_variance: float
K_means_model: int

Figura B.1: Estructura de parámetros del sistema

cada una. El valor asignado a estas variables queda indicado en la Tabla B.1.

- **MSBCv.normalized\_size**: Define el número de bytes que se utilizará para la función de escalado, expresión 3.2.
- **MSBCv.signature\_file**. Indica de que archivo se han de leer los modelos de cada aplicación, si de los generados por TIE, *own*, o si de los generados por una entidad externa, *foreign* (Apartado 4.4.2).

También se utiliza para el nombre del archivo encargado de asociar la identificación de cada aplicación dentro del fichero del que se han de leer los modelos, con los números identificativos usados por TIE.

- **MSBCv.calculate\_probability**. Este parámetro indica la forma en la que se calculará la probabilidad de que, dado un mensaje,  $s$ , este haya sido generado por  $g(n, l)$ , es decir, el modelado de una aplicación  $n$  para una posición en la secuencia de mensajes  $l$  (Apartado 4.21), pudiendo ser:
  - Mediante la suma de la probabilidad para cada uno de los clústeres, (*total*):
  - Tomada solo la probabilidad generada por aquella gaussiana con *centroide* mas próximo a la muestra (*max*):

Se selecciona un tipo u otro mediante *total* o *max* en el archivo de configuración *MSBC\_cont.txt*.

- **MSBCv.no\_gaussian\_available**. Este valor es usado si durante la clasificación no existen gaussianas estimadas para una posición de mensaje en una determinada dirección, asignando que la probabilidad de generar ese mensaje es el valor recogido por esta variable.
- **MSBCv.print\_map\_file**. Indica si se ha de imprimir el archivo *own\_map\_file.txt* durante la fase inicial del entrenamiento (Apartado 4.6.1).
- **MSBCv.num\_protocols\_train**. Indica el número de aplicaciones principales, esto es, el número de identificadores de TIE *app\_id* (Figura 4.13). .
- **MSBCv.clusters\_required**. Almacena el número de gaussianas máximo para el modelado de cada posición de secuencia de una aplicación,  $g(n, l)$ .
- **MSBCv.minimum\_error\_recoverable**. Mantiene el error mínimo que ha de recuperarse en un iteración del proceso de entrenamiento, tanto del bucle de bisección como del bucle clásico de K-medias, para que se puede ejecutar otra repetición.

Valores por defecto de cada parámetro		
Variable	Tipo	Valor
<i>MSBCv.normalized_size</i>	uint_16	512
<i>MSBCv.signature_file</i>	string	own
<i>MSBCv.calculate_probability</i>	int	2
<i>MSBCv.no_gaussian_available</i>	float	0.0
<i>MSBCv.print_map_file</i>	bool	<i>true</i>
<i>MSBCv.num_protocols_train</i>	uint_16	408
<i>MSBCv.clusters_required</i>	uint_16	15
<i>MSBCv.minimum_error_recoverable</i>	float	0.0000002
<i>MSBCv.euclidean_shift</i>	bool	<i>false</i>
<i>MSBCv.fixed_shift</i>	float	0.1
<i>MSBCv.step_modifier</i>	float	0.1
<i>MSBCv.euclidian_mean</i>	bool	<i>false</i>
<i>MSBCv.min_variance</i>	float	<i>false</i>
<i>MSBCv.K_means_model</i>	int	2

Tabla B.1: Valores por defecto para las variables globales del clasificador MSBC

Un valor alto de esta variable puede provocar que los clústres sean menos precisos, pero un valor muy pequeño puede alargar demasiado el proceso de agrupamiento innecesariamente.

- **MSBCv.euclidean\_shift.** Variable booleana que sirve para indicar si durante el proceso de bisección en el entrenamiento se debe aplicar la distancia euclídea o la distancia escalada. Si esta variable indica *true* se usará la distancia euclídea.
- **MSBCv.fixed\_shift:** Si se selecciona la posibilidad de la distancia euclídea en el proceso de bisección, esta variable indicará un desplazamiento fijo de los *centroides* generados, con respecto al *centroide* previo, en distancia euclídea.
- **MSBCv.step\_modifier.** Si se selecciona la posibilidad de la distancia escalada en el algoritmo de la bisección, este parámetro se usará para regular el desplazamiento entre los *centroides* (Apartado 3.4.3).

Con esta variable se regula el total del desplazamiento que vendrá dado en función de la desviación típica del clúster que se pretende dividir,  $\sigma$ .

- **MSBCv.euclidean\_mean.** Variable booleana que indica si se debe usar la distancia euclídea o la esclada para el cálculo del *centroide* de cada *clúster*. Si esta variable esta a *true* significa que debe aplicarse la distancia euclídea.

- **MSBCv.min\_variance.** Esta variable almacena la mínima varianza que debe tener el clúster más disperso (con una mayor varianza) durante el bucle de bisección de la fase de entrenamiento, para que no se de por concluido el proceso.
- **MSBCv.Kmeans\_model.** Indica el tipo de modelo para la bisección, (Apartado 4.6.3.2.5):
  - Se dividen todos los *clusters* (*square\_partition*)
  - Se divide solo el *clúster* más disperso (*one\_partition*)

## Apéndice C

# Procesado del tráfico externo

En el Apartado 4.6 se indicó que actualmente la posibilidad que ofrece TIE de generar sus propios ficheros de tráfico etiquetado o archivos *ground\_truth* no es demasiado eficaz, ya que los *plugins* con los que cuenta en esta versión o no son estables, como openDPI, o no ofrecen resultados fiables, como el clasificador basado en puertos o el clasificador conocido como *layer-7*.

Consecuentemente, para obtener dichos ficheros se aplicaron métodos alternativos. Por ello, se usaron ficheros en formato *pcap* cuyos flujos ya habían sido clasificados, quedando estos resultados almacenados en ficheros disponibles. En este apéndice se explican los procedimientos así como los programas usados con el objetivo de adaptar esos ficheros al formato que TIE utiliza como fichero de pre-clasificación o *ground truth*.

### C.1. Identificación de los protocolos presentes

#### C.1.1. Análisis de los campos útiles

En primer lugar se analizó detenidamente el formato de estos ficheros que guardan el etiquetado de los flujos, en busca de los elementos necesarios para el archivo de preclasificación como son las direcciones IP de origen y destino, los puertos de origen y destino, el protocolo de la capa de transporte y la aplicación que ha generado el flujo.

Un ejemplo del tipo de formato se muestra en la Figura C.1. En esta se comprueban las siguientes características del formato a procesar:

- Las aplicaciones vienen identificadas por una tupla formada por tres elementos, por ejemplo HTTP;HTTP;HTTP.
- Las direcciones IP origen y destino, hacen uso de términos *IP\_LOW* o *IP\_UPPER* donde no queda muy claro cómo se determinará el sentido de la comunicación, debido a que simplemente asigna un término u otro por orden numérico de las direcciones.

```
# Number of packets: [3508878] (194 fragmented or not IP)
# Format: BINARY OUTPUT (MEMORY DUMP)
FLOW_ID;FIRST_ID_PROT;LAST_ID_PROT;MAJ_ID_PROT;N_PROT;IP_LOW;IP_UPPER;PORT1;PORT2;
N_SIGNALING_DOWN;SHORT_PACKETS;SHORT_PACKETS_UP;SHORT_PACKETS_DOWN;LONG_PACKETS;
1;unknown;unknown;unknown;0;10.180.180.21;74.125.163.97;50040;80;TCP;DOWN;12754
2;HTTP;HTTP;HTTP;1;10.159.0.11;65.55.116.182;3052;80;TCP;UP;1275473232486720;1
3;unknown;unknown;unknown;0;10.145.0.107;213.199.149.108;1114;80;TCP;DOWN;12754
4;unknown;unknown;unknown;0;10.140.0.4;38.99.73.4;3629;80;TCP;DOWN;12754732324
5;unknown;unknown;unknown;0;10.193.0.59;213.199.149.108;49852;80;TCP;DOWN;12754
6;unknown;unknown;unknown;0;66.114.49.52;10.156.0.100;80;51979;TCP;DOWN;127547
7;unknown;unknown;unknown;0;10.156.150.75;120.28.232.161;50;60313;TCP;UP;12754
8;unknown;unknown;unknown;0;10.156.150.75;94.21.149.238;50;28797;TCP;UP;127547
9;unknown;unknown;unknown;0;10.159.0.11;65.55.81.103;3041;80;TCP;DOWN;12754732
10;HTTP;Flash;Flash;2;10.159.0.11;213.199.149.52;3050;80;TCP;UP;12754732326089
11;unknown;unknown;unknown;0;10.156.150.75;24.80.2.202;9666;34136;UDP;DOWN;127
12;unknown;unknown;unknown;0;66.114.49.52;10.156.0.100;80;51980;TCP;UP;1275473
13;unknown;unknown;unknown;0;10.156.150.75;76.174.53.132;50;63274;TCP;UP;12754
14;unknown;unknown;unknown;0;10.156.150.75;212.200.219.247;9666;48076;UDP;UP;1
15;unknown;unknown;unknown;0;10.156.150.75;217.66.22.83;9666;10813;UDP;UP;12754
16;unknown;unknown;unknown;0;10.156.150.75;122.177.231.185;9666;26552;UDP;UP;1
17;unknown;unknown;unknown;0;10.156.150.75;112.200.104.218;9666;28370;UDP;DOWN;
18;unknown;unknown;unknown;0;10.180.180.21;74.125.163.97;50034;80;TCP;DOWN;127
19;HTTP;HTTP;HTTP;1;10.159.0.11;64.208.138.216;3053;80;TCP;UP;1275473233452223;
```

Figura C.1: Fichero con el etiquetado de cada flujo del tráfico usado para el entrenamiento

- Los números de puerto origen y destino, hacen uso de términos *PORT1* o *PORT2*, donde *PORT1* estará asociado a *IP\_LOW* mientras que *PORT2* está con *IP\_UPPER*.
- Seguidamente, se indica el tipo de protocolo de la capa de transporte con el término *PORT*, pudiendo ser *TCP* o *UDP*.
- Finalmente, ya que los demás elementos no son relevantes para el fichero que se quiere implementar, se da la dirección *UP* o *DOWN*. Este término indica que:
  - Si es *UP*, la dirección establecida va del *IP\_LOW* al *IP\_UP*.
  - Si es *DOWN*, la dirección establecida va el *IP\_UP* al *IP\_LOW*.

Si se respeta la dirección establecida el mensaje enviado se considerará *upstream*, mientras que si no se respeta se tomará como *downstream*.

Dado que el sentido de la comunicación tiene un especial relevancia en el método aplicado, es necesario comprobar el criterio anterior. Para ello se tomará un flujo que tiene el término *UP*, esto indica que la comunicación se establece por primera vez desde la IP menor (*IP\_LOW*) hacia la IP mayor (*IP\_UPPER*). En este caso, se toma el flujo número dos, mediante *Wireshark* se muestra en la Figura C.2.

El primer paquete intercambiado será el que fijará el sentido de la conexión. Se ve como es la IP menor, 10.159.8.21, la que manda el primer paquete

Filter:	ip.addr == 65.55.116.182			▼	Expression...	Clear
No.	Time	Source	Destination	Protocol	TCP pa	
2	0.003383	10.159.0.11	65.55.116.182	TCP		
3	0.004668	10.159.0.11	65.55.116.182	HTTP		
36	0.480303	65.55.116.182	10.159.0.11	TCP		
▶Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)						
▶Ethernet II, Src: Cisco_00:22:6e (00:1c:f6:00:22:6e), Dst: D-Link_11:87:27 (00:						
▶Internet Protocol Version 4, Src: 10.159.0.11 (10.159.0.11), Dst: 65.55.116.182						
▼Transmission Control Protocol, Src Port: apc-3052 (3052), Dst Port: http (80),						

Figura C.2: Análisis de un intercambio de paquetes para un flujo del tráfico usado durante el entrenamiento

a la IP mayor, 65.55.16.182, por lo que el sentido *UP* es correcto, ya que la dirección *upstream*, que corresponde al primer paquete intercambiado, va de la IP pequeña a la mayor.

Una vez analizados los diferentes campos que son de utilidad, se observa que los que van a causar más problemas son los siguientes:

- Identificar las aplicaciones. Hay que asociar cada una de las aplicaciones en este fichero con sus números identificativos en TIE, *app\_id* y/o *sub\_id*.
- Hay que indicar la dirección IP de origen y destino entendiendo el tipo de asignación de dirección que realiza este fichero.
- Se debe realizar una asociación entre el protocolo de la capa de transporte y su número identificativo en TIE.

### C.1.2. Obtención de las aplicaciones presentes

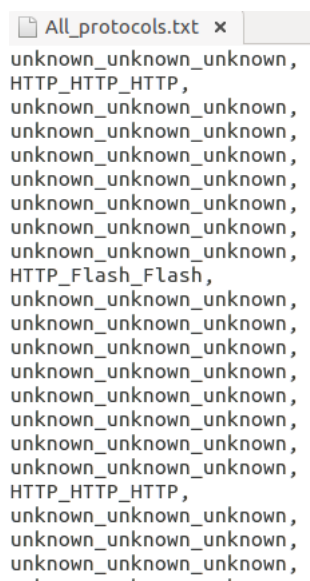
Con el propósito de tener un listado de todas las aplicaciones presentes, se genera un código en lenguaje en AWK que imprima todas las aplicaciones. Este código está almacenado en el fichero *Print\_all\_Protocols.awk* y la orden para ejecutarlo es la siguiente:

```
gawk -f Print_All_Protocols.awk param_db > All_protocols.txt
```

Donde *param\_db* es el fichero (C.1) que recoge los datos de los flujos a analizar, mientras que *All\_protocols.txt* almacenará el resultado de la orden anterior. El resultado de este proceso es lo que se observa en la Figura C.3, donde están las tuplas identificativas de cada aplicación presente en el fichero.

Posteriormente, se genera otro código, esta vez en C, que se encargará de tomar ese listado de tuplas identificativas y tomar solamente cada tupla una vez, evitando las repeticiones y permitiendo que el fichero de salida sea la base de una plantilla para asociar la identificación de TIE con estas tuplas.

Este código es el del archivo *No-repeated\_protocols.c*, que lo que hace es generar un vector de *strings* donde almacena las tuplas identificativas sin



```

All_protocols.txt x
unknown_unknown_unknown,
HTTP_HTTP_HTTP,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
HTTP_Flash_Flash,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
HTTP_HTTP_HTTP,
unknown_unknown_unknown,
unknown_unknown_unknown,
unknown_unknown_unknown,
...

```

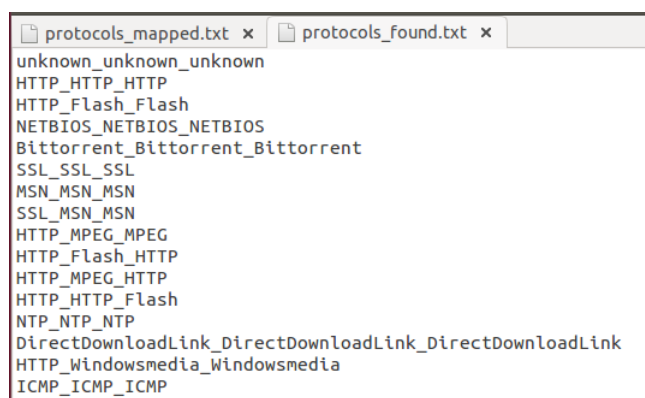
Figura C.3: Extracto del fichero All\_protocols.txt

repetir. Para ello, recorre el fichero *All\_protocols.txt* tomando aquellas tuplas que aún no se encuentren en el vector. Tras compilar el fichero, se ejecuta con la siguiente orden:

```
./No_repeated_protocols.o All_protocols.txt 100
```

Donde el primer argumento es el fichero a procesar y el segundo el tamaño del vector de *strings*, esto es, el número máximo de tuplas distintas que se pueden almacenar. Esta orden genera el fichero de la Figura C.4.

Tras esto, se deberá indicar manualmente la equivalencia entre los números identificativos de TIE y cada una de estas tuplas identificativas. El re-



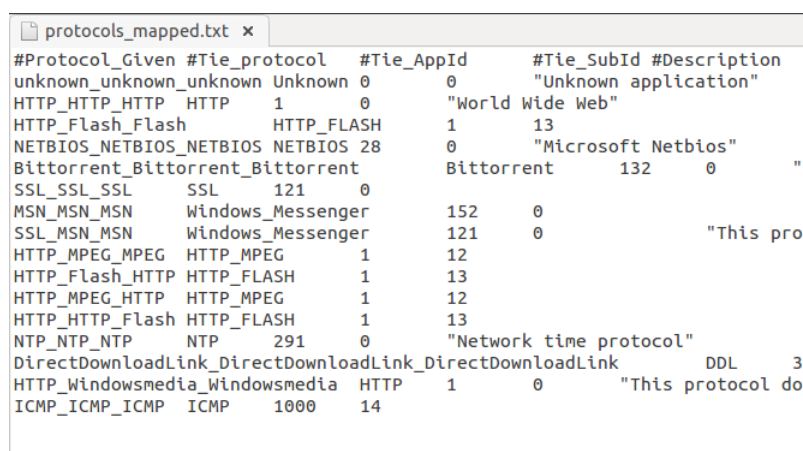
```

protocols_mapped.txt x protocols_found.txt x
unknown_unknown_unknown
HTTP_HTTP_HTTP
HTTP_Flash_Flash
NETBIOS_NETBIOS_NETBIOS
Bittorrent_Bittorrent_Bittorrent
SSL_SSL_SSL
MSN_MSN_MSN
SSL_MSN_MSN
HTTP_MPEG_MPEG
HTTP_Flash_HTTP
HTTP_MPEG_HTTP
HTTP_HTTP_Flash
NTP_NTP_NTP
DirectDownloadLink_DirectDownloadLink_DirectDownloadLink
HTTP_Windowsmedia_Windowsmedia
ICMP_ICMP_ICMP

```

Figura C.4: Contenido del fichero protocols\_found.txt





#Protocol_Given	#Tie_protocol	#Tie_AppId	#Tie_SubId	#Description
unknown_unknown_unknown	Unknown	0	0	"Unknown application"
HTTP_HTTP_HTTP	HTTP	1	0	"World Wide Web"
HTTP_Flash_Flash	HTTP_FLASH	1	13	
NETBIOS_NETBIOS_NETBIOS	NETBIOS	28	0	"Microsoft Netbios"
Bittorrent_Bittorrent_Bittorrent	Bittorrent		132	0
SSL_SSL_SSL	SSL	121	0	
MSN_MSN_MSN	Windows_Messenger	152	0	
SSL_MSN_MSN	Windows_Messenger	121	0	"This pro
HTTP_MPEG_MPEG	HTTP_MPEG	1	12	
HTTP_Flash_HTTP	HTTP_FLASH	1	13	
HTTP_MPEG_HTTP	HTTP_MPEG	1	12	
HTTP_HTTP_Flash	HTTP_FLASH	1	13	
NTP_NTP_NTP	NTP	291	0	"Network time protocol"
DirectDownloadLink_DirectDownloadLink_DirectDownloadLink	DDL			3
HTTP_Windowsmedia_Windowsmedia	HTTP	1	0	"This protocol do
ICMP_ICMP_ICMP	ICMP	1000	14	

Figura C.5: Contenido del fichero protocols\_mapped.txt

sultado de este proceso es el fichero de la Figura C.5.

Realizada esta tarea, termina la primera parte del proceso, ya que se dispone del fichero que servirá para asociar las diferentes tuplas identificativas de aplicaciones con sus números identificativos en TIE.

## C.2. Transformar al formato reconocido por TIE

Para llevar a cabo la adaptación del formato del fichero *param\_db*, de la figura C.1, se necesita recorrer cada una de las líneas que almacenan las características de un flujo, tomando cada uno de los parámetros y, posteriormente imprimiendo estos valores en otro archivo en el orden que TIE necesita y asociando aquellos campos con sus equivalentes en TIE, si es necesario.

En primer, lugar deberán de cargarse la asociación de identificadores de TIE con las tuplas usadas por el fichero. Para ello se genera una lista enlazada de estructuras como la de la Figura C.6, donde cada estructura se refiere a una de las aplicaciones reconocidas.

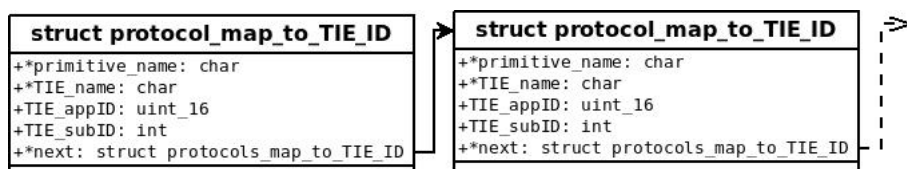


Figura C.6: Estructura de lista enlazadas usada para almacenar la asociación entre las identidades de TIE y las identidades usadas por el archivo de preclasificación externo

```

# begin TIE Table
# id      src_ip      dst_ip      proto      sport      dport      dwpkts
uppkts    dwbytes    upbytes    t_start    t_last     app_id     sub_id     confidence
1         74.125.163.97   10.180.180.21 6          80         50040      0
0         0          0          0          0          0          0          0
2         10.159.0.11     65.55.116.182 6          3052       80         0
0         0          0          0          0          1          0          0
3         213.199.149.108   10.145.0.107  6          80         1114       0
0         0          0          0          0          0          0          0
4         38.99.73.4        10.140.0.4    6          80         3629       0
0         0          0          0          0          0          0          0
5         213.199.149.108   10.193.0.59   6          80         49852      0
0         0          0          0          0          0          0          0
6         10.156.0.100        66.114.49.52  6          51979      80         0
0         0          0          0          0          0          0          0
7         10.156.150.75     120.28.232.161 6          50         60313      0
0         0          0          0          0          0          0          0

```

Figura C.7: Fichero ya transformado al formato de fichero de preclasificación reconocible por TIE

Cada una de estas estructuras almacenará la información siguiente acerca de la asociación de los elementos identificativos:

- Elemento *primitive\_name*. Tupla identificativa de la aplicación a la que hace referencia la estructura.
- Elementos *TIE\_appID* y *TIE\_subID*. Números identificativos de cada subaplicación en TIE.
- Elemento *\*next*. Indica la dirección de memoria de la siguiente estructura. Si no contiene ninguna dirección significa que no hay más estructuras.

Tras esto, se generará un bucle que recorrerá el fichero que se pretende pasar al formato reconocible por TIE. En cada iteración de este bucle se tomarán todos los elementos que se necesiten de un flujo, almacenándose en una serie de variables, para posteriormente ser imprimidas en el archivo de salida en el orden que TIE necesita.

En este proceso, además, se deberá identificar el equivalente de cada una de las tuplas en forma de número identificativo de TIE, haciendo uso de la lista enlazada antes expuesta.

Por otro lado, se deberá realizar algo similar a la hora de asociar el protocolo de transporte con su número identificativo en TIE. En este caso, solo se dispone de TCP y UDP ya que son los dos únicos que son necesarios, teniendo números identificativos de 6 y 17, respectivamente.

El programa generado para este fin se ejecuta con la siguiente orden:

```
./map_to_TIE_ground_file.c param_db protocols_mapped.txt
```

donde *param\_db* es el fichero con la preclasificación en el formato diferente al de TIE, mientras que *protocols\_mapped.txt* es el fichero generado manualmente que contiene la asociación de la identidad de cada aplicación en TIE, con su respectiva identidad en el archivo *param\_db*.

El resultado final de este proceso es el fichero de la Figura C.7



# Bibliografía

- [1] A. Callado, C. Kamienski, B. Gero :”*A Survey on Internet Traffic Identification and Classification*”. Publicado en: IEEE, ”*Communications Surveys and Tutorials*”, Vol: 3, pp: 37-52. 2009. DOI: 10.1109/SURV.2009.090304
- [2] A. Moore, K. Papagiannaki:”Toward the Accurate Identification of Network Applications”. Publicado en: ”*Passive and Active Network Measurement*”. Vol: 3431, pp: 41-54. 2005. DOI: 10.1007/978-3-540-31966-5\_4.
- [3] Amjad Hajjar, Jawad Khalife Jesús Díaz Verdejo. ”*Network Traffic Application Identification Based on Message Size Analysis*”
- [4] R. Haden: ”*Quality of Service*”. <http://www.rhyshaden.com/qos.htm>
- [5] William Stallings: ”*Network Security Essentials Applications and Standards*. Capítulo 9. 5ª edición. ISBN: 0133370437.
- [6] J. Reynolds: RFC 3232, ”*Assigned Numbers*”. <https://www.ietf.org/rfc/rfc1340.txt>. 2002
- [7] A. Dainotti, A. Pescapé, K.C.Claffy: ”*Issues and Future Directions in Traffic Classification*”. Publicado en: *Network*, IEE, vol.26, no.1, pp.35-40, 2012. DOI: 10.1109/MNET.2012.6135854.
- [8] A. Dainotti, F. Gargiulo, L.I. Kuncheva, A. Pescapé, C. Sansone: ”*Identification of Traffic Flows Hiding TCP Port 80*”. Publicado en: ”*IEEE International Conference on Communications (ICC), 2010*”. pp.1-16,23-27, 2010. DOI: 10.1019/ICC.2010.5502266.
- [9] G. Aceto, A. Dainotti, W. de Donato, A. Pescapé : ”*PortLoad: Taking the Best of Two Worlds in Traffic Classification*”. Publicado en: ”*INFOCOM IEEE Conference on Computer Workshops*. Conferencia. pp 1-5. 2010. DOI: 10.1109/INFCOMW.2010.546645.
- [10] T. Nguyen, G. Armitage: *A Survey of techniques for internet traffic classification using machine learning*. Publicado en: ”*Communi-*

- cations Surveys and Tutorials*", IEEE, vol. 10,nº 4,pp. 56-76.2008. DOI: 10.1109/SURV.2008.080406
- [11] M. Jaber, R.G.Cascella, C. Barakat: "*Can We Trust the Inter-Packet Time for Traffic Classification*". Publicado en: "*Communications (ICC), 2011 IEEE International Conference on*". pp. 1-5. 2011. DOI: 10.1109/icc.2011.5963024.
- [12] L. Bernaille, R. Teixeira, K. Salamatian: "*Early Application Identification*". Publicado en: "*Proceedings of the 2006 ACM CoNEXT Conference*". se. CoNEXT'06. pp. 6:1–6:12. 2006. DOI: 10.1145/1368436.1368445.
- [13] W. de Donato, A. Pescapè, A. Dainotti: "*Traffic Identification engine: an open platform for traffic classification*". Publicado en: "*Network*", IEEE, vol.28, no.2, pp.56-64, 2014. DOI: 10.1109/M-NET.2014.6786614.
- [14] Anil K. Jain: "*Data clustering: 50 years beyond k-means*". Publicado en: "*Pattern Recognition letters*", vol.31, nº8, pp. 651-666, 2010. DOI: 10.1026/j.patrec.2009.09.011.
- [15] H. Gupta, A. Agraval: "*Global K-means (GKM) Clustering Algorithm: A survey*". Publicado en: "*International Journal of Computer Applications(0975-8887)*". vol. 79, nº2, 2013. pp. 20-24. DOI: 10.5120/13713-1472.
- [16] *Berkeley Packet Filter (BPF) syntax*. Url: <http://biot.com/capstats/bpf.html>
- [17] Biblioteca *libpcap* de diferentes tipos de flujos. *WireShark Samples Captures*. <https://wiki.wireshark.org/SampleCaptures>