Programa Oficial de Posgrado en Sistemas Multimedia

Departamento de Teoría de la Señal, Telemática y Comunicaciones

Universidad de Granada

TESIS DOCTORAL

Protección de redes P2P mediante análisis de tráfico

Realizada por:

Rafael A. Rodríguez Gómez

Dirigida por:

Dr. Pedro García Teodoro

Dr. Gabriel Maciá Fernández

Editor: Editorial de la Universidad de Granada
Autor: Rafael A. Rodríguez Gómez
D.L.: GR 1847-2014
ISBN: 978-84-9083-030-7

Official Postgraduate Program in Multimedia Systems

Department of Signal Theory, Telematics and Communications

University of Granada

Ph.D. Thesis Dissertation:

P2P network protection through traffic analysis

Author:

Rafael A. Rodríguez Gómez

Advisors:

Dr. Pedro García Teodoro

Dr. Gabriel Maciá Fernández

El doctorando D. Rafael A. Rodríguez Gómez y los directores de la tesis Dr. D. Pedro García Teodoro y Dr. D. Gabriel Maciá Fernández, catedrático y profesor titular de universidad respectivamente y adscritos al Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada,

GARANTIZAMOS AL FIRMAR ESTA TESIS DOCTORAL

que el trabajo ha sido realizado por el doctorando bajo la tutela de los directores de la tesis y, hasta donde nuestro conocimiento alcanza, en la realización del trabajo se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, a 9 de enero de 2014

Directores de la Tesis

Dr. D. Pedro García Teodoro

Dr. D. Gabriel Maciá Fernández

Doctorando

D. Rafael A. Rodríguez Gómez

Agradecimientos

Sé el cambio que deseas ver en el mundo

Gandhi

Permitidme la licencia de expresarme con libertad (y escasa formalidad) en estas pocas líneas.

Creo que es éste un buen momento para parar el ritmo frenético al que nos vemos abocados y decir GRACIAS. No un gracias de compromiso, ni un "gracias" esperando recibir un "de nada", sino un gracias por todo lo que se me ha regalado.

¡Gracias!

La sensación final que me queda después de este periodo predoctoral es precisamente de gratitud. Tal vez sea cierto lo que se barrunta en el ambiente de que la universidad está podrida y que su investigación no sirve a la sociedad y no tiene aplicación práctica, o que los que la componen no son trabajadores ni se preocupan por los alumnos. Doy las gracias al departamento de Teoría de la Señal, Telemática y Comunicaciones por su personal, mis compañeros, con ejemplos claros de lo contrario, preocupados tanto por la mejora de la sociedad con su investigación como por los alumnos y su formación. Aún hoy pienso, gracias a ellos, que es posible que la universidad sea un espacio de formación en el que la búsqueda de conocimiento sea emocionante y que el trabajo en ella se viva como un servicio a la sociedad, ¡gracias NESG!

Mirando estos cuatro años me inundan buenos recuerdo pero sobretodo, lo que más viene a mi mente y lo que más valoro, han sido las relaciones personales que hemos creado y mantenido. Gracias a los que estáis y me habéis acompañado, compañeros de los distintos despachos por los que he pasado (Moria, Orquídea, CITIC) y alumnos que me han ayudado a ilusionarme por el mundo de la docencia. En especial a José, transmites ilusión y eso es un gran don que admiro en ti; Carlos, por mucho que lo intentes nunca parecerás un tipo duro, ¡eres un buenazo!; Santi, gracias por estar siempre al servicio pero dedícate tiempo a ti mismo que ya te va tocando ◡̈; Juanjo, siempre lo haces todo para que nadie te lo agradezca, GRACIAS me has dado mucho; Celso, eres la persona más inteligente que conozco y sabes cuidar de los demás como nadie (¡tienes mucho que dar!); Carlos Alarcón, por tus conversaciones profundas y pausadas; Héctor, porque no dejas que nos separemos. Por supuesto no podían faltar Lluvia, Alberto, Rita, Alextoni, Sponja, Roberio Tagán, Leovigildo Sandez *and*, *of course*, *our friend* Joagoiu T. Valdeszaucas y muchos más a los que me gustaría dar las gracias pero ya son más de las 12 y temo convertirme en calabaza ;)

Quiero agradecer de forma especial a Gabri y a Pedro, mis directores de tesis. Siempre habéis confiado en mis posibilidades, me habéis apoyado de forma constante y gracias, principalmente a vosotros, siento que he adquirido un gran capacidad de trabajo autónomo, búsqueda de problemas (de los buenos) y de soluciones a los mismos.

Gracias también a los amigos selectos que se han colado en mi vida sin querer y, como no, a Nacho, Manu y Juan porque sois mi válvula de escape.

Gracias a mi familia porque mi visión de la vida es culpa vuestra y porque me queréis como soy. Gracias Noe por enseñarme lo verdaderamente importante de la vida y por ser el ejemplo de gratuidad y sencillez, ¡MUCHO!

A todos vosotros y a los demás os digo: ¡SED FELICES!

Resumen

Hoy en día es innegable la importancia de Internet, la *Red de redes*, en multitud de las interacciones diarias de una gran parte de la población a nivel mundial. Gracias a Internet es posible actualmente utilizar una gran cantidad de servicios que permiten a los usuarios comunicarse sin ningún esfuerzo con otras personas de cualquier parte del mundo. Entre ellos se pueden destacar el servicio web, el correo electrónico o la compartición de recursos mediante P2P (*Peer-to-Peer*), por mencionar algunos.

El primer modelo de comunicación utilizado en Internet fue el modelo cliente-servidor, sencillo de implementar pero con un gran problema de escalabilidad, que conlleva que al aumentar el número de usuarios la calidad del servicio ofrecido disminuye. Como solución a este problema de escalabilidad apareció el modelo P2P. Los ejemplos de redes más significativas en las que se utiliza este modelo son las llamadas redes P2P, como eDonkey, BitTorrent, Spotify o Skype, entre otras. Estas redes representan actualmente un porcentaje considerable de todo el ancho de banda mundial. Sin embargo, este nuevo modelo de comunicaciones posee también un claro inconveniente, y es que presenta multitud de vulnerabilidades y amenazas a la seguridad. La naturaleza de la propia filosofía P2P implica que no existe ningún servidor centralizado que se encargue de subir, almacenar y comprobar la autenticidad de los recursos. Una consecuencia directa de ello es que no hay ningún mecanismo para controlar qué contenido se comparte. Esto hace que las redes P2P sean óptimas para la propagación de *malware* en general.

En este contexto, el objetivo fundamental del presente trabajo de tesis es *mejorar la seguridad de las redes P2P*. Para esto se propone una aproximación en dos pasos: (*i*) diseñar e implementar sistemas de *identificación de tráfico P2P* y (*ii*) aplicar medidas de defensa concretas ante ataques a la seguridad en estas redes.

La clasificación de tráfico de red es de gran utilidad para que los *Internet Service Provider*s (ISPs) conozcan el tráfico que viaja por sus redes y, a partir de ello, sean capaces de aplicar de forma diferenciada políticas de seguridad en función del tráfico identificado. De esta forma, será posible afrontar el tema de la seguridad en redes P2P de una forma específica para el tráfico de este tipo. En esta línea, en el presente trabajo se presentan una breve revisión bibliográfica de la clasificación de tráfico de red, dos algoritmos de detección de tráfico eDonkey y un sistema de clasificación de tráfico P2P basado en modelado de Markov.

Tras un estudio de los ataques de ciberseguridad en redes P2P actuales se muestra que una de las principales amenazas de la ciberseguridad de nuestros días son las redes de *zombies* o *botnets*. Por ello, primeramente se ha realizado una revisión de la bibliografía más relevante de los últimos años en este tema, organizándola en base a una novedosa taxonomía aquí propuesta. En segundo lugar se ha diseñado y

evaluado un sistema de detección de *botnets* P2P parásitas. Este sistema de detección se basa en el comportamiento de compartición de los recursos de la botnet en lugar de en las comunicaciones de red de ésta, lo que constituye un hecho diferencial respecto de otras soluciones en este campo existentes en la bibliografía.

Abstract

Nowadays, Internet, *the Network of networks*, is present in many of the daily normal interactions of a great part of the world. Thanks to the Internet is possible to use multiple services that allow users to communicate between them without effort. Some example of these services are web, e-mail or services to share resources through P2P (*Peer-to-Peer*) networks, among others.

The first communication model used in the Internet was the client-server model. The implementation of its communications is simple but it is not scalable, *i.e.*, if the number of users grows the quality of the offered service will decrease. As a possible solution to the scalability problem the P2P model appeared. Some examples of the use of this communication model are eDonkey, BitTorrent, Spotify or Skype, among others. P2P networks currently represent a high percentage of the total world's bandwidth. However, this new communication model has a clear disadvantage: it presents several vulnerabilities and security threats. In the P2P model every node of the network can be a client or a server and, this way, there is none responsible of uploading, saving and checking the authenticity of the shared resources. As a consequence, there are no mechanisms to control the contents shared in the network which makes P2P networks ideal to propagate malware.

In this context, the main objective of the present thesis work is *to improve the security of P2P networks*. To do this we propose an approach in two steps.

The fist step is to design efficient and effective *P2P traffic identification* mechanisms. *Internet Service Provider*s (ISPs) need to know the type of the traffic that travels through their networks in order to apply security policies depending on the detected traffic, and this is done by means of traffic classification. Thus, they would be able to face P2P network security policies in a specific manner for this type of traffic. In this line in the present work a brief bibliographic review is presented, as well as two detection algorithms for eDonkey protocol, and a P2P traffic classification system based on hidden Markov models.

The second step consists of applying specific particular defense measures against attacks to the security in P2P networks. After a study of the current cibersecurity attacks in P2P networks, it is shown that one of the main current threats are the so-called zombie networks or *botnets*. Thus, firstly a review of the most relevant works in the last years in the field of botnets is presented, organizing these works following a new taxonomy proposed here. Secondly, a detection system for P2P parasite botnets has been designed and evaluated. This detection approach represents an innovation regarding previous botnet detection schemes because it is based on the temporal evolution of shared resources between bots instead of on network communications among them.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Abreviaturas y Acrónimos

AHP *Analytic Hierarchy Process*

ARPANET *Advanced Research Projects Agency Network*

CUSUM *CUMulative SUM*

C&C *Command and Control*

DA *Detection Accuracy*

DDoS *Distributed Denial Of Service*

DDNS *Dynamic Domain Name System*

DGA *Domain Generation Algorithm*

DHT *Distributed Hash Table*

DIY *Do-it-Yourself*

DNS *Domain Name System*

DNSBL *DNS-based Block List*

DoS *Denial Of Service*

DPI *Deep Packet Inspection*

EFFORT *Efficient and Effective Bot Malware Detection*

FBI *Federal Bureau of Investigation*

FN *False Negative*

FP *False Positive*

FPR *False Positive Rate*

FPS *First-Person Shooter*

FTP *File Transfer Protocol*

HMM *Hidden Markov Model*

HTTP *Hypertext Transfer Protocol*

HTTPS *Hypertext Transfer Protocol Secure*

IANA *Internet Assigned Numbers Authority*

IDS *Intrusion Detection System*

IRC *Internet Relay Chat*

ISP *Internet Service Provider*

KCFM *Kalman filter and multi-chart CUSUM Fused Model*

KL *Kullback Leibler*

LTE *Long Term Evolution*

MTU *Maximum Transfer Unit*

NAT *Network Address Translation*

NPV *Negative Predictive Value*

PLC *Programmable Logic Controller*

PPI *Pay-Per-Install*

PPM *Passive P2P Monitor*

PPV *Positive Predictive Value*

P2P *Peer-to-Peer*

RDNS *Recursive Domain Name System*

ROC *Receiver Operating Characteristic*

RTP *Real-time Transport Protocol*

SCADA *Supervisory Control And Data Acquisition*

SSH *Secure SHell*

SSL *Secure Sockets Layer*

STRIDE *System to Retrieve Information from Drug Evidence*

SMTP *Simple Mail Transfer Protocol*

SNMP *Simple Network Management Protocol*

TCP *Transmission Control Protocol*

TN *True Negative*

TP *True Positive*

TPR *True Positive Rate*

TTL *Time To Live*

UDP *User Datagram Protocol*

URND *Up Rate-based Node Detection*

VoIP *Voice over IP*

WCFD *Way Changed-based Flow Detection*

Capítulo 1

Introducción

La época actual es conocida como la era de las comunicaciones y una de las razones que ha motivado este apelativo es que es posible comunicarse de forma sencilla e instantáneamente con personas de todo el mundo. Esto es factible gracias a la impresionante evolución de Internet, que apareció en el año 1969 de la mano de ARPANET (*Advanced Research Projects Agency Network*) con únicamente dos ordenadores interconectados y en la que se comunican ahora miles de millones de equipos.

El paradigma en el que se basaron las primeras aplicaciones de Internet, y en el que se siguen sustentando servicios tan comunes como el de la web o el correo electrónico, es el paradigma cliente-servidor (ver Figura 1.1(a)). En este modelo de comunicación, una aplicación instalada en el cliente interactúa con otra que se ejecuta en el servidor. El cliente es la entidad encargada de iniciar la comunicación hacia el servidor enviando una solicitud, mientras que éste último debe permanecer a la escucha de nuevas peticiones. Tras ello, el servidor se encarga de procesar la solicitud del cliente y enviar de vuelta una respuesta.

Una de las ventajas principales de este paradigma es que resulta sencillo implementar las comunicaciones implicadas. Sin embargo, la mayor desventaja es su reducida escalabilidad. Es complicado mantener el nivel de servicio que puede ofrecer un servidor cuando crece sustancialmente el número de clientes que acceden al mismo. De los servicios basados en este paradigma se pueden destacar el servicio web, el correo electrónico o la transferencia de ficheros por FTP (*File Transfer Protocol*) o SSH (*Secure SHell*), entre otros muchos.

Con posterioridad a la aparición del paradigma anterior, y de la mano del incremento en la capacidad de ancho de banda de las conexiones de los usuarios de

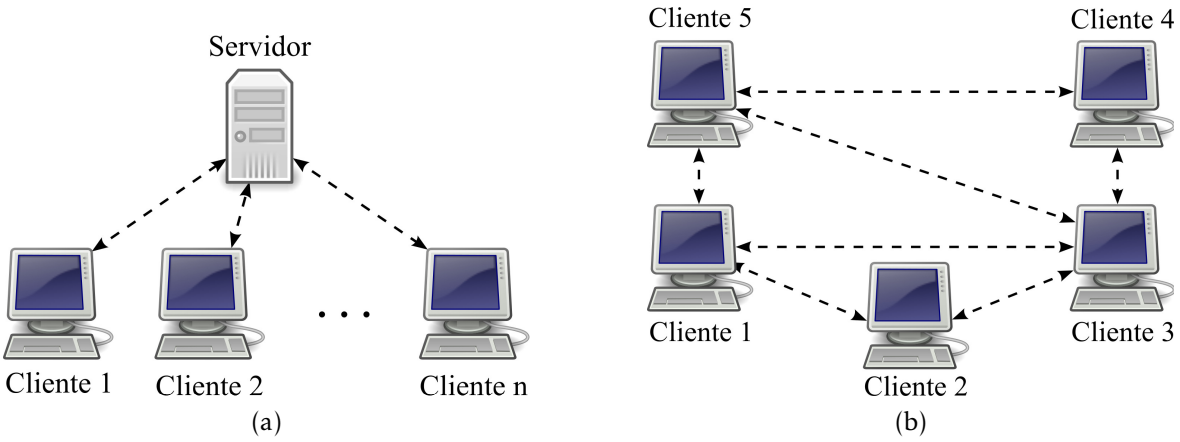


Figura 1.1: Esquema de los paradigmas (a) cliente-servidor y (b) P2P.

Internet, surge el modelo P2P (ver Figura 1.1(b)). En este nuevo paradigma cada entidad tiene capacidad para actuar como cliente y también como servidor. De este modo, es posible que una entidad inicie comunicaciones con otras, pero también que reciba solicitudes entrantes. Por tanto, podemos entender que todas las entidades de este modelo desempeñan un mismo papel, por lo que son denominadas pares o *peers*.

En el modelo P2P es posible gestionar el problema de la escalabilidad de forma mucho más eficaz que en el modelo cliente-servidor, gracias a la posibilidad de distribuir la ejecución de un determinado servicio entre los múltiples *peers* de la red. Sin embargo, la gestión de la red es más compleja, siendo necesario, por ejemplo, establecer mecanismos para identificar dónde se pueden encontrar los servicios, desarrollar protocolos específicos para las comunicaciones entre *peers*, etc. Esto suele derivar en redes menos eficientes que las basadas en el modelo tradicional en relación al tiempo necesario para enviar una solicitud y recibir la respuesta correspondiente.

Muchos de los servicios que originalmente seguían el modelo cliente-servidor poseen su análogo en la actualidad en el modelo P2P. Un ejemplo claro de esta evolución puede ser la transferencia de archivos, ahora llevada a cabo, en múltiples escenarios, mediante redes P2P como BitTorrent o eDonkey. Incluso es posible encontrar aplicaciones que requieren una gran eficiencia en sus comunicaciones como son los juegos de tipo FPS (*First-Person Shooter*) que también se basan en redes P2P, este es el ejemplo del conocido juego *Halo* [1]. Así, el uso actual de estas redes en Internet está ampliamente extendido y, como muestra este último ejemplo, se prevé que su expansión será aún mayor. Es notable también la relevancia de estas redes debido a que el volumen de tráfico derivado del uso de las mismas supone un porcentaje sustancial de todo el ancho de banda global [2].

Sin embargo, este nuevo modelo de comunicaciones posee también un claro inconveniente, y es que estas redes presentan multitud de vulnerabilidades y ame-

nazas a la seguridad. La naturaleza de la propia filosofía P2P implica que no existe ningún servidor centralizado que se encargue de subir, almacenar y comprobar la autenticidad de los recursos. Una consecuencia directa de ello es que no hay ningún mecanismo para controlar qué contenido se comparte. Esto hace que las redes P2P sean óptimas para la propagación de *malware* en general. Otro ejemplo de los posibles usos maliciosos de estas redes reside en el hecho de que poseen, de forma distribuida, una gran capacidad de subida y descarga que puede ser aprovechada [3] para realizar ataques de denegación de servicio distribuidos.

La seguridad en redes es un tema de relevancia especial en la era de las comunicaciones, recurrente en la investigación y es frecuente encontrar noticias relacionadas con la misma. Pero, ¿es realmente tan importante? ¿Cuál es su magnitud? Un dato que puede arrojar luz acerca de la importancia de este problema es un estudio publicado en 2011 por Norton Symantec [4]:

Con 431 millones de víctimas globales en el pasado año, y con unas pérdidas mundiales de 388.000 millones de dólares, el cibercrimen ha pasado a representar ampliamente una mayor suma de dinero que el que mueven los mercados negros de marihuana, cocaína y heroína juntos, que suponen unos 288.000 millones de dólares.

Derivado de la importancia creciente del tema de la seguridad en redes en general, el objetivo fundamental de este trabajo de tesis es *mejorar la seguridad de las redes P2P*. Para esto se propone una aproximación en dos pasos.

El primer paso es diseñar mecanismos eficaces y eficientes de **identificación de tráfico P2P**. La clasificación de tráfico [5] es el proceso automático por el cual se categoriza el tráfico de red en distintas clases en función de varios parámetros, como pueden ser el puerto de la comunicación, las cabeceras de los paquetes, características del flujo, etc. Es de una gran utilidad que los *Internet Service Provider*s (ISPs) conozcan el tráfico que viaja por sus redes para, de esta forma, ser capaces de aplicar de forma diferenciada políticas de seguridad, o incluso sistemas de detección y respuesta de anomalías en función del tráfico identificado. Por este motivo, la clasificación de tráfico es actualmente un problema muy popular en el campo de las redes de telecomunicación, y concretamente entre la comunidad investigadora.

Una vez abordado el problema de la clasificación de tráfico P2P, el segundo paso de la aproximación propuesta en este trabajo consiste en aplicar medidas de defensa concretas ante ataques a la seguridad en estas redes. Tras un estudio de los ataques de ciberseguridad en redes P2P actuales, se ha identificado que una de las principales amenazas de la ciberseguridad de nuestros días son las redes de *zombies* o *botnets*. El término *botnet* se utiliza para definir una red de máquinas infectadas, llamadas *bots*, que son controladas por un operador humano, comúnmente conocido como *botmaster*. Los *bots* se utilizan para una gran variedad de actividades maliciosas y dañinas contra sistemas y servicios, incluyendo ataques DoS (*Denial Of Service*), distribución de *spam*, distribución de *malware*, *phishing* y fraude del *click* [6] [7].

El peligro creciente que suponen las *botnets* ha sido apuntado por muchos autores en distintos trabajos, como puede ser el caso de Vinton Cerf, uno de los "padres de Internet", que estimó que entre 100 y 150 de los 600 millones de equipos conectados a Internet formaban parte de una *botnet* [8]; o un estudio de Symantec de 2007 [9], en el que se detectó una media de 61.940 *bots* activos por día; o en un estudio posterior de McAfee en 2009 [10], en el que se descubrieron cerca de 12 millones de nuevas direcciones IP actuando como *bots*, lo cual supone un incremento notable con respecto a las cifras aportadas por Symantec.

Este problema requiere de un **estudio** detallado de la bibliografía actual y una **detección** concreta con respecto a las amenazas aún sin resolver en el campo de las *botnets*, objetivo perseguido por el presente trabajo.

1.1. Objetivos

El objetivo fundamental del presente trabajo de tesis es mejorar la seguridad de las redes P2P. Este objetivo general se ha divido en dos objetivos particulares: (*i*) clasificación de tráfico P2P y (*ii*) estudio y detección de *botnets* P2P. A continuación se exponen las tareas específicas asociadas a cada uno de ellos:

I. *Identificación de tráfico P2P*

- Diseñar un sistema de identificación de tráfico P2P eficiente y eficaz.
- Implementar físicamente el sistema diseñado.
- Evaluar su correcto funcionamiento por medio de una experimentación realista.

II. *Estudio y detección de* botnets

- Estudiar el estado del arte de la investigación en el campo de las *botnets*.
- Estructurar de una forma organizada e intuitiva los trabajos de investigación estudiados.
- Diseñar un esquema de detección frente a las *botnets* P2P parásitas, amenaza especialmente dañina y aún sin propuestas concretas de defensa en la actualidad.
- Evaluar el esquema de detección propuesto en un escenario realista.

1.2. Contribuciones principales

Las contribuciones fundamentales de este trabajo de tesis pueden resumirse en los siguientes puntos:

1. Se presenta una breve revisión bibliográfica del campo de la clasificación de tráfico de red.

2. Se contribuye con dos algoritmos de detección de tráfico eDonkey. Uno de ellos para la detección de flujos de este protocolo, WCFD (*Way Changed-based Flow Detection*), y el segundo de detección de nodos generadores de tráfico eDonkey, URND (*Up Rate-based Node Detection*).

3. Se diseña e implementa un sistema de clasificación de tráfico basado en modelos ocultos de Markov que, con un esfuerzo reducido, puede ser utilizado para detectar múltiples protocolos diferentes. Se realiza la particularización de este sistema para la detección de tráfico del protocolo eDonkey.

4. Se diseña una taxonomía basada en el ciclo de vida de las *botnets* que permite agrupar de forma organizada la investigación realizada en este campo.

5. Se contribuye con una revisión de la bibliografía más relevante de los últimos años en el tema de las *botnets*, organizándola en base a la taxonomía anterior.

6. Se presenta un sistema de detección de una de las amenazas emergentes más importantes relativas al campo de las *botnets*: las *botnets* P2P parásitas. Este sistema de detección representa una innovación en cuanto a las detecciones de *botnets* anteriores, ya que se basa en la evolución temporal de los recursos compartidos por éstas y no en las comunicaciones de red de las mismas.

1.2.1. Publicaciones

Las publicaciones derivadas de este período de tesis doctoral y directamente relacionadas con el trabajo aquí expuesto se indican a continuación:

Revistas internacionales

1. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, and P. García-Teodoro. 2013. "Survey and taxonomy of *botnet* research through life-cycle". *ACM Comput. Surv.* 45, 4, Article 45 (August 2013), 33 pages.

2. *Enviada* – > **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, P. García-Teodoro, M. Steiner, and D. Balzarotti. 2013. "Resource monitoring for the detection of parasite P2P botnets". *Computer Networks*.

3. *Seleccionada para ser enviada* – > **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, and P. García-Teodoro. 2013. "Analysis and monitoring of resources shared in BitTorrent Network". *Transactions on Emerging Telecommunications Technologies*.

Congresos internacionales

4. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Stochastic Traffic Identification for Security Management: eDonkey Protocol as a Case Study". *7th Network and System Security* (NSS 2013), pp. 1-13, 2013.

5. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Analysis of *Botnets* through Life-cycle". Proceedings of *SECRYPT*, pp. 257-262, 2011.

6. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "New Heuristics for eDonkey Node and Flow Detection". *P2P Systems* (AP2PS), pp. 90-95, 2011.

Congresos nacionales

7. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Monitorización y Análisis de los Recursos Compartidos en la Red BitTorrent". *XI Jornadas de Ingeniería Telemática* (JITEL 2013), pp. 243-248.

8. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Acceso a servicios basado en modelado de Markov: eDonkey como caso de estudio". *Reunión Española sobre Criptología y Seguridad de la Información* (RECSI 2012), pp. 51-56.

9. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Nuevas heurísticas para la detección de nodos y flujos eDonkey". *X Jornadas de Ingeniería Telemática* (JITEL 2011), pp. 276-283.

Existen otras publicaciones que no han sido fruto directo del presente trabajo pero que sí han formado parte del período predoctoral. Éstas se indican a continuación.

Revistas internacionales

a) G. Maciá-Fernández, Y. Wang, **R. A. Rodríguez-Gómez** and A. Kuzmanovic. "Extracting User Web Browsing Patterns from Non-Content Network Traces: The Online Advertising Case Study", *Computer Networks*, Vol. 56. 2012, pp. 598-614.

b) G. Maciá-Fernández, **R. A. Rodríguez-Gómez** and J. E. Díaz-Verdejo. "Defense techniques for low-rate DoS attacks against application servers", *Computer Networks*, Vol. 54, October, 2010, pp. 2711-2727.

Congresos internacionales

c) L. Sánchez-Casado, **R. A. Rodríguez-Gómez**, R. Magán-Carrión and G. Maciá-Fernández. "NETA: Evaluating the effects of NETwork Attacks. MANETs as a case study". *Advances in Security of Information and Communication Networks*, (SecNet 2013), pp. 1-10.

d) G. Maciá-Fernández, Y. Wang, **R. A. Rodríguez-Gómez** and A. Kuzmanovic. "ISP-enabled behavioral ad targeting without deep packet inspection". Proceedings of the *29th conference on Information communications*, (INFOCOM 2010), pp. 1-9.

Congresos nacionales

e) L. Sánchez-Casado, **R. A. Rodríguez-Gómez**, R. Magán-Carrión and G. Maciá-Fernández. "NETA: un Framework para Simular y Evaluar Ataques en Redes Heterogéneas. MANETs como Caso de Estudio". *XI Jornadas de Ingeniería Telemática* (JITEL 2013), pp. 501-506.

f) **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and J. E. Díaz-Verdejo. "Defensas frente a ataques DoS a baja tasa contra servidores basadas en políticas de gestión de colas". *VIII Jornadas de Ingeniería Telemática* (JITEL 2009), pp. 46-53.

Libros

g) G. Maciá Fernández, R. Magán Carrión, **R. A. Rodríguez Gómez**, L. Sánchez Casado: "Sistemas y Servicios Telemáticos". 2013. Ed. Avicam. ISBN: 978-84-941781-6-0.

1.3. Estructura del documento

Este documento, de acuerdo a lo ya expuesto para los objetivos con anterioridad descritos, ha sido dividido en dos partes fundamentales, ambas en la línea de una seguridad integral de las redes P2P. Es importante destacar que se ha hecho un esfuerzo de redacción para que los capítulos sean autocontenidos, de modo que un lector interesado sólo en uno de ellos pueda comprenderlo en sus líneas básicas sin necesidad de leer los capítulos previos.

1.3.1. Primera parte: Identificación de tráfico P2P

La conceptos fundamentales de clasificación de tráfico, así como una breve revisión bibliográfica de este campo, se exponen en el Capítulo 2.

En el Capítulo 3 se describen dos algoritmos de detección para la identificación de tráfico eDonkey, protocolo de compartición de recursos mediante P2P. Uno de estos algoritmos tiene como objeto detectar los flujos eDonkey (WCFD) y el otro los nodos generadores de tráfico correspondiente al mencionado protocolo (URND).

En el Capítulo 4 se detalla un sistema de clasificación de tráfico de red basado en el uso de modelos ocultos de Markov. A diferencia de la clasificación propuesta en el Capítulo 3, este sistema puede ser ampliado con facilidad para detectar múltiples protocolos.

1.3.2. Segunda parte: Estudio y detección de *botnets*

En esta segunda parte se aborda uno de los temas más críticos de la ciberseguridad de nuestros días: las *botnets*. Inicialmente, en el Capítulo 5, se exponen los conceptos fundamentales relacionados con las *botnets*, se presenta una nueva taxonomía de *botnets* basada en el ciclo de vida de las mismas y se realiza una revisión bibliográfica de los artículos más relevantes de este campo, agrupándolos en base a la taxonomía propuesta.

Una de las nuevas amenazas que se ponen de manifiesto en el estudio de *botnets* son un tipo concreto de *botnets* cuyas comunicaciones se sustentan en redes P2P ya existentes, de modo que sus comunicaciones quedan ocultas dentro del tráfico legítimo de estas redes. Estas *botnets* se denominan *botnets* P2P parásitas. En el Capítulo 6 se propone y evalúa un sistema de detección para este tipo de *botnets* basado en la evolución temporal de la compartición de los recursos.

Por último, en el Capítulo 7 se exponen las principales conclusiones que se derivan del trabajo completo, resaltando algunas líneas de trabajo futuro que han

de abordarse para continuar avanzando en la línea de investigación iniciada con la presente tesis doctoral.

Apéndices

Adicionalmente, a las dos partes mencionadas, con objeto de cumplir con la normativa vigente relativa a las tesis con mención internacional, se incluyen dos apéndices en inglés al final de este documento. El primero de ellos, Apéndice A, presenta un resumen amplio del documento completo que expone de forma sintética los trabajos realizados tanto en la primera parte de clasificación de tráfico como en la segunda de estudio y detección de *botnets*, ambas partes motivadas por el objetivo global del presente documento, mejorar la seguridad de las redes P2P. En el Apéndice B se indican las conclusiones y líneas de trabajos futuros ya presentadas en el Capítulo 7.

Parte I

Identificación de tráfico P2P

Capítulo 2

Clasificación de tráfico de red: Estado del arte

Durante los últimos años la caracterización y el modelado de tráfico de red han constituido, y aún constituyen, un aspecto relevante en el campo de las redes de comunicación. En este sentido, se han llevado a cabo multitud de esfuerzos con el objetivo de caracterizar y, subsecuentemente, identificar y/o clasificar este tipo de tráfico. Entre las aplicaciones fundamentales de la clasificación de tráfico de red se destacan: (*i*) el modelado del uso y las dinámicas de las aplicaciones de red, (*ii*) la identificación de forma temprana de las nuevas aplicaciones, (*iii*) la detección de posibles comportamientos anómalos y (*iv*) la tarificación de los servicios ofrecidos por los proveedores de servicio.

En este capítulo se presentan los conceptos fundamentales de la clasificación de tráfico de red y un breve repaso de algunos de los trabajos más relevantes en este campo, poniendo un énfasis especial en la clasificación de tráfico P2P por ser ésta el objetivo fundamental de los próximos dos capítulos. La revisión bibliográfica presentada se divide en tres partes en función de la fuente de información utilizada en la clasificación: métodos basados en puertos, métodos basados en paquetes y métodos basados en flujos.

El resto del capítulo se estructura de la siguiente forma. En la Sección 2.1 se motiva la necesidad de la clasificación de tráfico de red en la actualidad. Los principales conceptos sobre clasificación de tráfico de red se clarifican en la Sección 2.2. Seguidamente, una breve revisión bibliográfica de algunos de los trabajos más relevantes del campo, con especial énfasis a la clasificación de tráfico P2P, es presentada en la Sección 2.3. Por último, las conclusiones principales del presente capítulo se exponen en la Sección 2.4.

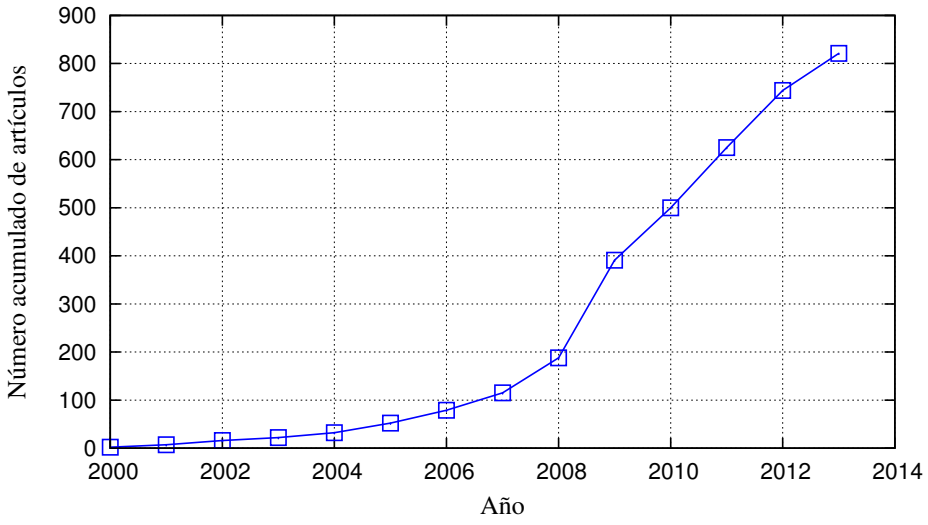


Figura 2.1: Suma acumulada del número de publicaciones en el campo de la clasificación de tráfico de red en los últimos años (datos tomados del portal Scopus [11]).

2.1. Motivación

La caracterización y clasificación del tráfico de Internet se ha convertido en uno de los retos más populares en el campo de las redes de telecomunicación en los últimos años. Como puede verse en la Figura 2.1, el elevado número de publicaciones en el campo de la clasificación de tráfico de red, y su continuo crecimiento en los últimos años, evidencian la importancia de este tema entre la comunidad investigadora. Los ISPs precisan un amplio conocimiento de la composición y las dinámicas del tráfico de Internet para gestionarlo de forma eficiente. En general, la caracterización del tráfico de Internet facilita multitud de tareas de gestión de red, como son: planificación de la capacidad, ingeniería de tráfico, eficiencia de las aplicaciones de red, detección de anomalías, etc.

Adicionalmente, la disponibilidad de conexiones de banda ancha y la capacidad de éstas se encuentra en continuo crecimiento, llegando ya a muchos hogares el cable de fibra óptica, que multiplica la capacidad de acceso a la red hasta hoy disponible para los usuarios. Este incremento del ancho de banda ha derivado en un nuevo uso de Internet tanto en usuarios particulares como en pequeñas y medianas empresas. Actualmente, los usuarios utilizan asiduamente servicios disponibles en la Internet de hoy como son: VoIP, comercio electrónico, banca electrónica y aplicaciones P2P de compartición de recursos (principalmente se comparten recursos de vídeo o de audio), entre otros servicios. Esto, por tanto, ha derivado en un aumento de la complejidad del uso de la red por parte un usuario normal [12]. Los ISPs deben prestar más atención a este comportamiento complejo, especialmente de cara a la popularización de las nuevas tecnologías de acceso inalámbrico, entre las que se encuentran 4G LTE

(*Long Term Evolution*) y Wi-Max [13], tecnologías que a su vez también incrementan sustancialmente el ancho de banda del que dispone el usuario final.

Algunos estudios recientes destinados a analizar el tráfico de Internet [2] indican que en nuestros días el tráfico generado por las aplicaciones de compartición de recursos mediante P2P representa, dependiendo de la región, entre un 43 % y un 70 % de todo el ancho de banda de Internet. Estos estudios son complejos y presentan limitaciones, como es la duración limitada de la monitorización o las pérdidas de información durante el proceso de medida. Pero es innegable que el volumen de tráfico derivado del uso de aplicaciones basadas en redes P2P es uno de los principales responsables del elevado consumo de ancho de banda en las redes actuales.

Con el objetivo de identificar el tipo de aplicaciones de red que se usan en la actualidad y el impacto que éstas presentan sobre la red es importante disponer de técnicas de clasificación de tráfico eficaces y eficientes. Ésta es una de las razones que han motivado a la comunidad investigadora a trabajar en este tema, derivando ello, como ya se ha indicado a través de la Figura 2.1, en una gran cantidad de trabajos de investigación. Por ello, en este primer capítulo se presenta una breve revisión del estado del arte de clasificación de tráfico, con un énfasis especial en la clasificación de tráfico P2P.

2.2. Conceptos básicos sobre clasificación de tráfico de red

La clasificación de tráfico de red [5] es el proceso automático por el cual el tráfico de red es categorizado en diferentes clases en función de multitud de parámetros, como por ejemplo el puerto de comunicación, las cabeceras de los paquetes, ciertas características de los flujos de red, etc. En esta sección se presentan algunos de los conceptos básicos sobre el proceso de la clasificación de tráfico de red. En primer lugar, se indican cuáles son las aplicaciones fundamentales que motivan el estudio y uso de este tipo de técnicas; seguidamente se describen algunos de los módulos comunes presentes en el proceso de clasificación; y por último se presentan algunas de las medidas más extendidas para evaluar la bondad de los sistemas de clasificación de tráfico de red.

2.2.1. Aplicaciones de la clasificación de tráfico de red

La clasificación de tráfico de red es aplicada con múltiples objetivos. Atendiendo a los trabajos [14] [15] se destacan los siguientes cuatro casos en los que su empleo es fundamental:

- *Identificación del uso y de las dinámicas de las aplicaciones de red*

  La clasificación de tráfico de red aplicada a la identificación de aplicaciones de usuario y sus dinámicas es de gran utilidad para los operadores de red y los proveedores de servicio. Con esta información, los operadores de red pueden aplicar métodos de ingeniería de tráfico para el dimensionado y planificación de sus redes, y los ISPs por su parte pueden ofrecer distintas calidades de servicio en función de la demanda de los usuarios.

- *Identificación de aplicaciones nuevas*

  La clasificación temprana del uso de aplicaciones de red nuevas puede arrojar algo de luz sobre la influencia de éstas en las dinámicas del tráfico de red. De esta forma, se podrá evitar que algunos servicios importantes puedan verse afectados, o incluso que su uso llegue a ser denegado como consecuencia de la creciente implantación de estas nuevas aplicaciones cuyo alcance es aún desconocido.

- *Detección de anomalías*

  Disponer de mecanismos de detección de anomalías es de crucial importancia tanto para los usuarios finales como para los operadores de red, con el objetivo de incrementar la seguridad de los datos de red y la disponibilidad de los servicios ofertados. Un ejemplo de anomalía en el tráfico de red puede ser la provocada por un ataque DDoS (*Distributed Denial Of Service*) ejecutado por una *botnet*. La anomalía consistiría en un incremento enorme del nivel de tráfico de red percibido por el servidor objetivo del ataque.

- *Tarificación de servicios*

  Desde la perspectiva de un ISP resulta de gran importancia conocer las aplicaciones de red que utilizan sus clientes de cara a la tarificación de ciertos servicios, o incluso para ofrecer nuevos productos. Por ejemplo, un ISP podría estar interesado en conocer los usuarios que están utilizando VoIP porque en el contrato de su servicio de 3G no permite las llamadas mediante esta tecnología.

2.2.2. Evaluación de los sistemas de clasificación de tráfico de red

Para evaluar la calidad de un sistema de clasificación de tráfico es necesario realizar una comparación de los resultados que éste devuelve aplicándolo a tráfico de red real, es decir, es necesario analizar los resultados del mismo. Esta validación podría llevarse a cabo a través de un procedimiento de inspección manual en el que un experto analiza los datos y los categoriza en las clases correspondientes. El inconveniente de este proceso es que no puede realizarse con un elevado número de datos ya que requeriría demasiado tiempo por parte del experto. Adicionalmente, un proceso manual resulta tedioso y está sujeto a la posible comisión de errores. Un

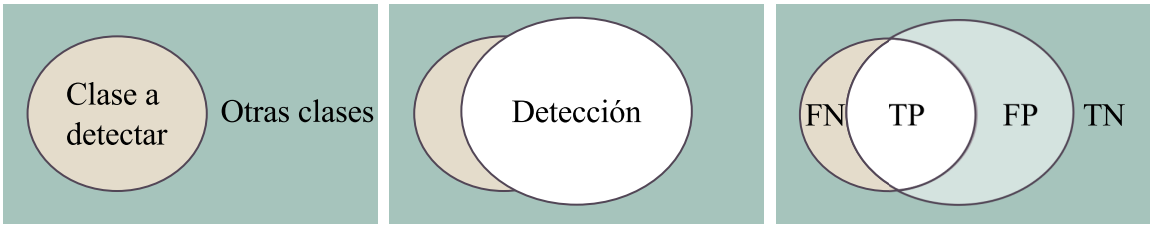


Figura 2.2: Medidas de la bondad de un proceso de clasificación.

procedimiento alternativo y habitual en la validación del proceso de clasificación de trazas de tráfico de red es el uso de herramientas de inspección de paquetes y la asunción de este resultado como cierto. Las técnicas de inspección de paquetes consisten en inspeccionar en detalle los contenidos de los paquetes de red para identificar patrones que permitan asociar dichos paquetes a una aplicación concreta. A esta técnica se la denomina comúnmente DPI (*Deep Packet Inspection*). Una aplicación muy utilizada para este fin es OpenDPI [16].

Sea cual sea el método de validación utilizado, manual o automático, el resultado es un conjunto de datos etiquetados indicando la clase (protocolo o servicio) a la que pertenecen. Este conjunto de datos etiquetados se denomina *ground truth* y es considerado como la referencia con respecto a la que comparar los resultados del sistema de clasificación a evaluar.

Para llevar a cabo la validación del sistema de clasificación objeto de análisis primero se procederá a su aplicación sobre un conjunto de datos dispuestos a tal efecto, de manera que éstos se etiquetarán como pertenecientes a la clase a detectar (*p.ej.*, un protocolo de comunicación concreto) o a cualquier otra clase. Esto es lo que se muestra en la Figura 2.2(a). Los resultados arrojados por el sistema de clasificación se representan en la Figura 2.2(b), los cuales evidencian clasificaciones válidas y no válidas en el sentido de que algunas de las trazas identificadas corresponderán a la clase a detectar, mientras que otras habrán sido etiquetadas de forma incorrecta. Finalmente, en la Figura 2.2(c) se representa la correspondencia entre los conjuntos de datos y las medidas más comunes para indicar la calidad de los resultados de clasificación. Éstas son las siguientes, y se derivan de la clasificación realizada y el conocimiento de la *ground truth*:

- TP (*True Positive*). Son los datos detectados por el sistema de clasificación que según la *ground truth* pertenecen realmente a la clase a detectar.
- FP (*False Positive*). Éstos son datos clasificados como tráfico de la clase a detectar pero que realmente no pertenecen a ella. Esto es, FP se refiere a una clasificación incorrecta por parte de nuestro sistema.

- TN (*True Negative*). Los datos que no son detectados como pertenecientes a la clase a detectar y que realmente no pertenecen a ella.
- FN (*False Negative*). Los datos de la clase a detectar que no son detectados por el sistema de clasificación. Como FP, FN se refiere a una clasificación errónea.

En base a estas cuatro medidas se definen las siguientes tasas:

- Tasa de verdaderos positivos, TPR (*True Positive Rate*). También conocida como sensibilidad, *recall* o exhaustividad:

$$TPR = \frac{TP}{TP + FN} \tag{2.1}$$

- Tasa de falsos positivos, FPR (*False Positive Rate*), o *fall-out*:

$$FPR = \frac{FP}{FP + TN} \tag{2.2}$$

- Valor negativo predicho, NPV (*Negative Predictive Value*):

$$NPV = \frac{TN}{TN + FN} \tag{2.3}$$

- Valor positivo predicho, PPV (*Positive Predictive Value*), o precisión:

$$PPV = \frac{TP}{TP + FP} \tag{2.4}$$

- Exactitud, *accuracy*:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.5}$$

Una representación muy común acerca de la calidad de un sistema de clasificación es la curva ROC (*Receiver Operating Characteristic*). Esta curva representa la evolución de las tasas de FP y de TP en función de los parámetros del sistema de clasificación. Su uso permite mostrar la robustez del sistema frente a pequeños cambios de sus parámetros. El uso fundamental de la curva ROC es escoger el punto de operación óptimo para el que el sistema es capaz de arrojar mejores resultados.

En la Figura 2.3 se muestra un ejemplo de la curva ROC ideal de un sistema de clasificación. Esta curva ROC indica que el sistema de clasificación ideal es aquel que no presenta ningún FP, y que a la vez que detecta correctamente todos los datos de la clase objetivo, independientemente de los parámetros del sistema que

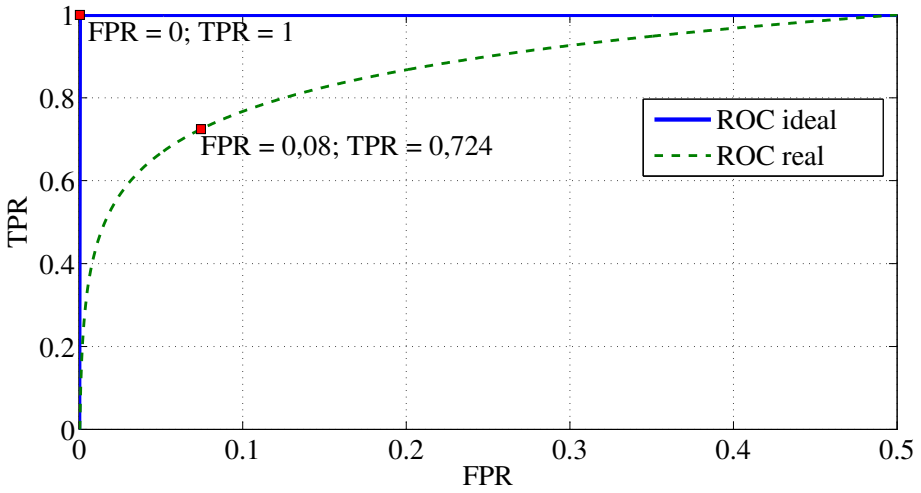


Figura 2.3: ROC ideal y ejemplo de una ROC real de un sistema de clasificación.

se escojan. El punto de operación escogido para este sistema de clasificación sería el marcado con un cuadrado, que supone un TPR del 100 % y un FPR del 0 %. Como contrapunto, en la misma figura se muestra una ROC más habitual propia de un sistema de clasificación real, donde FPR y TPR no son siempre los deseados y los falsos positivos han de ser tenidos en cuenta para afrontarlos de forma conveniente.

Para extraer las métricas anteriores con respecto a un sistema de clasificación de tráfico es necesario disponer de conjuntos de trazas de tráfico etiquetado, de modo que exista una relación entre el tráfico presente en las mismas y los protocolos a los que pertenece. Estos conjuntos de trazas de tráfico se dividen comúnmente en tres:

1. *Conjunto de entrenamiento*

   El primer paso para poder utilizar un sistema de clasificación de tráfico es realizar un entrenamiento con el conjunto de trazas de entrenamiento. Estas trazas suelen contener únicamente tráfico del protocolo que pretende ser detectado y se utilizan para hallar los parámetros propios que modelan el mismo.

2. *Conjunto de ajuste*

   Una vez que el sistema ha sido entrenado es necesario *ajustar* los parámetros del mismo para su uso en un entorno real. Los parámetros que se obtienen con este conjunto de trazas suelen ser los relativos a la identificación del tráfico, como por ejemplo los valores de los umbrales que indican si la salida del sistema ha de interpretarse como perteneciente al protocolo a detectar o no. Por tanto, este conjunto contiene múltiples protocolos incluyendo el protocolo a detectar.

   En sistemas de clasificación sencillos es común realizar el ajuste y el entrenamiento en un sólo paso y utilizando, por tanto, un mismo conjunto de datos.

3. *Conjunto de validación*

   Por último, una vez que el sistema de clasificación ha sido entrenado y ajustado se utiliza, para la validación, este conjunto de trazas de tráfico que ha de ser totalmente diferente de los conjuntos de entrenamiento y ajuste. Es con este conjunto de trazas con el que se caracteriza el sistema, generalmente hallando su curva ROC.

2.2.3. Proceso típico de clasificación de tráfico de red

Según los trabajos disponibles en la literatura especializada, tres son los pasos o etapas fundamentales a considerar en un sistema de clasificación de tráfico:

- *Parametrización del tráfico de red*

  Se han estudiado y propuesto en la literatura multitud de características para representar el tráfico de red como paso previo a la clasificación. Un ejemplo puede ser el uso de las cabeceras de TCP (*Transmission Control Protocol*) y los primeros bytes del campo de datos de los paquetes [17]. En otros trabajos se propone una caracterización múltiple; por ejemplo, los autores de [18] consideran el número de paquetes ARP, la velocidad media, el tamaño medio del paquete, la proporción de tráfico TCP/UDP y la duración de la comunicación para representar el tráfico observado.

- *Proceso de clasificación*

  Los esquemas utilizados para llevar a cabo la clasificación en sí misma cubren un amplio rango de técnicas. Desde simples heurísticas o indicadores, como en [19] [20] [21], hasta complejas técnicas de minería de datos o algoritmos de aprendizaje de patrones, como en [22] [23] [24]. Además, algunos autores proponen combinar más de un clasificador, como ocurre en [25].

- *Nivel de identificación*

  En la literatura se consideran principalmente dos niveles para llevar a cabo el proceso de clasificación[26] [27], nivel de identificación de flujos o de paquetes. En el caso de la identificación de flujos el objetivo es clasificar cada flujo como perteneciente o no a un servicio dado. Por otro lado, en la identificación de paquetes el objetivo es clasificar cada paquete individualmente. Lo usual es mezclar ambos niveles de identificación como sucede en [27].

2.3. Revisión bibliográfica de las propuestas sobre clasificación de tráfico de red

Seguidamente presentamos un breve estudio del estado del arte de los métodos de clasificación de tráfico más relevantes hasta la fecha, dividiéndolos en tres grupos: (*i*) basados en el puerto, (*ii*) basados en paquetes y (*iii*) basados en flujos. Se pondrá especial énfasis en la clasificación de tráfico P2P por ser éste un objetivo fundamental de la presente tesis.

2.3.1. Métodos de clasificación basados en puertos

El método de clasificación de tráfico más simple utilizado consiste en aprovechar la correspondencia establecida por la IANA (*Internet Assigned Numbers Authority*) [28] entre aplicaciones y el número de puerto reservado para ellas. Entre las correspondencias más conocidas se pueden encontrar las siguientes: HTTP (*Hypertext Transfer Protocol*) asociado al puerto 80, DNS (*Domain Name System*) al 53, los puertos 20 y 21 se reservan para FTP, y el correo electrónico implica el uso de puertos como 25, 110 o 143.

La principal ventaja de este método es que resulta muy poco costoso computacionalmente y que sólo requiere el análisis de la información de las cabeceras de los paquetes. La implementación concreta de un sistema de clasificación basado en puertos consiste en almacenar las correspondencias conocidas entre puertos y sus aplicaciones asociadas. Su extensión es también muy sencilla, ya que en el momento en el que aparezca una nueva aplicación de red bastará con añadir al sistema una nueva correspondencia entre esta aplicación y el puerto asociado a ella para que todo siga funcionando perfectamente.

Hace ya algunos años que este método de clasificación de tráfico ha sido descartado por ser considerado un método ineficaz [29] [30] [31]. Ello se debe a la propia evolución de las aplicaciones de red, que en la actualidad pueden asignar sus puertos de forma dinámica en lugar de consultarlos de forma estática en las correspondencias establecidas por la IANA. Una motivación que sustenta esta asignación dinámica es la de ocultar las comunicaciones dentro de los puertos comunes de otros protocolos de red. Un ejemplo de ello es la utilización del puerto 80, asociado a HTTP, que suele estar permitido por la mayoría de los cortafuegos. Múltiples aplicaciones como pueden ser las de compartición de recursos mediante P2P, BitTorrent o eDonkey, utilizan esta aproximación de asignación dinámica de puerto, lo que ha forzado a la comunidad investigadora a desarrollar nuevos métodos de clasificación de tráfico que no estén basados en la mera identificación del puerto de escucha.

2.3.2. Métodos de clasificación basados en paquetes

Los métodos de clasificación basados en paquetes toman información a nivel de paquete para realizar la clasificación. Es importante notar que un método de clasificación puede estar basado en el procesado de paquetes pero que su identificación sea a nivel de flujos, es decir, la información utilizada para la clasificación proviene de los paquetes pero el objetivo de clasificación son los flujos del protocolo y no los paquetes.

El método de clasificación más eficaz es el basado en paquetes y, más concretamente, el basado en el análisis completo de todos los campos del paquete para identificar el protocolo con el que se ha generado. Conociendo a la perfección el protocolo y analizando todos los campos de su contenido se puede identificar con bastante seguridad el protocolo que generó el paquete. Esto es lo que llevan a cabo aplicaciones tan conocidas como OpenDPI o Wireshark. Sin embargo, este método de clasificación también presenta ciertas deficiencias, *p.ej.*, algunos protocolos cifran sus comunicaciones por motivos de seguridad, impidiendo de esta forma el análisis del contenido de sus paquetes. Otro problema asociado a este tipo de clasificación es el de los protocolos propietarios, cuya descripción no es pública. Así, al no disponer de una descripción de los estados y mensajes propios del protocolo, no es posible explorar los paquetes en busca de unos patrones que realmente no se conocen.

El método de clasificación por paquetes es poco escalable, ya que no es posible analizar de forma completa todos los paquetes de todos los flujos de tráfico de red para determinar el protocolo o servicio concreto, de entre todos los conocidos, que los generó. Adicionalmente, ampliar la capacidad de clasificación ante la aparición de un nuevo protocolo o servicio no es nada sencillo, ya que requiere de un enorme trabajo de implementación. Por último, una gran cantidad de países poseen leyes que prohíben a los operadores de red acceder al contenido de los paquetes que viajan por sus redes, por ser una violación de la privacidad.

Todas las dificultades anteriormente comentadas indican que el análisis completo de los protocolos no es una solución en sí misma, aunque puede ser utilizada en combinación con métodos menos complejos computacionalmente y que sean escalables. Una posible aproximación a la clasificación de tráfico menos costosa computacionalmente es la de buscar patrones de bytes concretos en el contenido de los paquetes. Estos patrones se denominan *firmas*. Así por ejemplo, el tráfico web contiene la cadena `GET` y los paquetes del protocolo eDonkey comienzan por los bytes `xe3`. La característica común de estos métodos de clasificación de tráfico basados en contenido (o basados en firmas) es que buscan tanto en las cabeceras de los paquetes como en el contenido de éstos. Este tipo de métodos suele presentar una tasa de falsos positivos muy reducida, ya que no es común que otro protocolo presente firmas específicas de un protocolo. Sin embargo, también es difícil elevar la sensibilidad (tasa de verdaderos positivos) de los mismos. Un ejemplo de esta aproximación de

clasificación en el campo del tráfico P2P es [32] en el que los autores diseñan una metodología para, tras el análisis concreto del protocolo a detectar, extraer reglas alimentan un IDS (*Intrusion Detection System*) con objeto de identificar el tráfico correspondiente a aplicaciones como: WinMx o KaZaA.

Sen et al. [29] presentan una propuesta en la que analizan las aplicaciones P2P más comunes utilizando firmas de la capa de aplicación. Su aproximación es capaz de obtener en la mayoría de los casos, inspeccionando solamente los primeros 10 paquetes del flujo, menos de un 5 % de falsos positivos y falsos negativos. Sin embargo, si el flujo viaja encriptado esta solución no sería factible. Adicionalmente, como se comentó anteriormente, muchos países prohíben la inspección del contenido de los paquetes de red impidiendo que esta aproximación pueda ser utilizada a nivel internacional y, consecuentemente, válida.

Existen otras propuestas que se basan en analizar las características estadísticas a nivel de paquetes de red para realizar la clasificación del tráfico. Por ejemplo, variaciones abruptas en la tasa de generación de paquetes de un equipo podrían deberse a la propagación de una *botnet*. Sin embargo, estas variaciones también podrían ser consecuencia del inicio de una aplicación de compartición de recursos en el equipo escaneado. En suma, la capacidad de clasificación de esta aproximación es generalmente pobre.

Es común observar que en lugar de clasificar aplicaciones o servicios concretos se intente diferenciar un tipo de tráfico de otro, *p.ej.*, tráfico P2P de tráfico web. En [33] los autores realizan una identificación estadística de los tipos de aplicaciones dividiéndolos en 4 categorías: interactivo, transferencia de datos, *streaming* y transacional. Con objeto de alcanzar mejores resultados de detección, los métodos estadísticos suelen combinarse con técnicas de inteligencia artificial. El método más utilizado es el Bayesiano, como en [15] y [34]. Otros trabajos, como [35] y [36], hacen uso de esquemas de *machine learning*.

El trabajo de Moore et al. [15] utiliza análisis Bayesiano para clasificar flujos. Para realizar el proceso de clasificación es necesario disponer de bases de datos de entrenamiento y de test. La base de datos de entrenamiento ha de estar etiquetada y será considerada la *ground truth* que permitirá concluir la bondad del sistema de clasificación. El principal problema de esta aproximación es que requiere de un gran número de flujos etiquetados para crear la *ground truth*. Los resultados de clasificación obtenidos son notablemente buenos, a excepción del caso de aplicaciones de compartición de archivos mediante redes P2P, en el que sólo alcanzan un 36,45 % de aciertos. Además, este método no puede ser utilizado en tiempo real, con la clara limitación que ello implica.

Bernaille et al. [37] [38] [39] realizan una clasificación del tráfico basándose sólo en las cabeceras de los paquetes. Estos métodos de clasificación sólo necesitan los primeros paquetes de cada flujo. En [38] utilizaron solamente los primeros 5 paquetes

(con contenido de aplicación) de una conexión en cada dirección. La exactitud para el caso de un protocolo de compartición de recursos mediante P2P como es eDonkey es relativamente baja, concretamente del 84,2%.

Se han propuesto algunas otras aproximaciones diferentes en la literatura pero ninguna de ellas es capaz de clasificar todos los distintos tipos de tráfico de aplicación presentes en Internet. Por tanto, aparecen métodos con el objetivo de aunar las bondades de varios métodos existentes, como es el caso de [40]. En este trabajo la clasificación se basa en los resultados individuales de múltiples métodos de clasificación, llevándose a cabo la clasificación final a partir de un complejo mecanismo de decisión. Respecto a las prestaciones de este tipo de sistemas de clasificación, hemos de mencionar que la cantidad de tráfico sin clasificar disminuye enormemente y la confianza de la clasificación aumenta, ya que los distintos métodos validan los resultados entre sí.

También existen aportaciones destinadas a la clasificación de un único protocolo. Como ejemplo de este tipo de clasificación se puede citar la detección de tráfico Skype [41] [42], en la que los autores proponen un sistema basado en dos técnicas complementarias para detectar en tiempo real el tráfico perteneciente a dicho protocolo. La primera aproximación se basa en el test Chi-Cuadrado de Pearson y en las características del tráfico de VoIP para detectar, en la estructura de los paquetes, huellas fundamentales del tráfico Skype, valiéndose de la aleatoriedad introducida a nivel de bit en el proceso de encriptación. El segundo proceso se basa en la caracterización estadística del tráfico Skype en términos de la tasa de llegada y el tamaño de los paquetes.

Uso de modelado de Markov para clasificar tráfico de red basado en paquetes

No son muchas las contribuciones existentes en la bibliografía que utilicen modelos de Markov, pero éstas presentan una relevancia especial para el presente trabajo ya que el sistema de clasificación propuesto en el marco de la presente tesis doctoral en el Capítulo 4 se basa en modelos de Markov. Por este motivo a continuación se indican algunos de los trabajos más relevantes propuestos hasta la fecha.

Los autores en [43] utilizan modelos de Markov para representar el comportamiento de un flujo específico, utilizando para ello la información de los paquetes de control, *p.ej.* SYN, ACK, SYN-ACK, PSH-ACK, PSH, etc. Esta aproximación requiere de un número elevado de paquetes de control para obtener buenos resultados de detección. Incluso con un número alto los resultados obtenidos presentan una elevada tasa de falsos positivos; por ejemplo, con 50 paquetes de control en un flujo existe un 10% de falsos positivos al intentar diferenciar entre HTTP y HTTPS (*Hypertext Transfer Protocol Secure*).

Otra propuesta en este campo es la contribución de Wright et al. [44], donde se sigue un diseño similar al utilizado en el alineamiento de secuencias de proteínas. Los autores utilizan un HMM (*Hidden Markov Model*) de izquierda a derecha con un número de estados igual al número medio de paquetes con contenido en los flujos del servicio a detectar. Los resultados obtenidos son muy variables dependiendo del servicio objetivo; de 58,20 % a 92,90 % para la tasa de aciertos y de 7,90 % a 0,62 % para la de falsos positivos.

En [37] los autores utilizan también HMMs y comparan sus resultados con otras técnicas de clasificación. El objetivo de su trabajo es clasificar de forma temprana servicios basados en TCP, utilizando para ello entre los 4 y 10 primeros paquetes de cada flujo. Una gran desventaja de esta aproximación es que sólo pueden detectar protocolos basados en TCP. Además, alcanzan tasas de acierto bastante reducidas, estando éstas en torno al 70 % para el protocolo eDonkey.

El modelo propuesto en [45] por Dainotti et al. presenta un modelo diferente para cada servicio a detectar. Un punto débil de esta propuesta es el reducido tamaño de las bases de datos de tráfico utilizadas. Por ejemplo, para el tráfico del juego *Age of Mitology* se utilizan 4 flujos para entrenar y 2 para evaluar; por tanto, los resultados obtenidos no son representativos en absoluto.

2.3.3. Métodos de clasificación basados en flujos

Un flujo se define, según [46], como el tráfico identificado por la tupla `<IP-origen,puerto-origen,IP-destino,puerto-destino,protocolo>`, donde origen y destino son intercambiables para permitir tráfico bidireccional. Este tipo de métodos utilizan como fuente de información los flujos de tráfico.

La clasificación de tráfico basada en datos de nivel de flujo suele proporcionar unas prestaciones inferiores que la clasificación basada en paquetes, ya que los datos considerados contienen menos información. La metodología principal utilizada en la literatura se basa en un análisis estadístico.

Existe una gran cantidad de trabajos en los que se propone una clasificación basada en las características de los flujos. BLINC [14] es un trabajo importante que presenta una aproximación novedosa a la clasificación de flujos de tráfico. Esta herramienta de clasificación asocia un equipo final con la aplicación que genera la mayor parte de su tráfico. Es decir, asocia los equipos finales con los servicios que ofrecen o utilizan, en lugar de analizar cada flujo individualmente. En esencia, BLINC analiza los patrones de conexión de un nodo en tres niveles de detalle: (*i*) social, (*ii*) funcional y (*iii*) de nivel de aplicación. En el social se utilizan las direcciones IP para analizar las comunicaciones de un equipo con otros. En el nivel funcional se incluye la información de puertos de destino y de origen, identificándose los nodos como

servidores si reciben muchas conexiones a un mismo puerto desde puertos diferentes, como clientes en el caso contrario y como colaborativos en un caso intermedio. Por último, en el nivel de aplicación se clasifica el nodo como generador de un tipo de aplicación en función de la relación entre las direcciones y los puertos origen y destino, junto con la duración y el número de bytes intercambiados en ambas direcciones de la conexión.

También es especialmente relevante el trabajo de Xu et al. en [47], en el que se presenta una metodología para crear perfiles de comportamiento del tráfico de Internet en términos de patrones de comunicación de usuarios finales y servicios. En primer lugar, los datos se agregan en cinco grupos de flujos y estos flujos a su vez se agrupan en *clusters*. En el siguiente paso, se extrae el *cluster* con mayor significado. Esto se hace computando el tamaño del *cluster* y comprobando si los *clusters* que son clasificados como "sin significado" aparentan tener un comportamiento aleatorio. En el tercer paso, se clasifica el comportamiento del *cluster* dentro de una de las clases de comportamiento. Finalmente, las clases de comportamiento se asocian con aplicaciones conocidas.

Otros trabajos, en lugar de utilizar una única metodología para clasificar todos los protocolos existentes, clasifican únicamente un subconjunto de protocolos. Esta es la aproximación más frecuente para clasificar flujos de los protocolos utilizados en redes P2P. En esta línea se encuentra [31], el primer trabajo que intenta clasificar tráfico proveniente de aplicaciones P2P en puertos arbitrarios sin inspeccionar el cuerpo de los paquetes. Para esto se utilizan dos heurísticas: *(i)* en la primera se seleccionan las parejas IP origen y destino que se comunican utilizando tanto TCP como UDP (*User Datagram Protocol*), y *(ii)* la segunda se basa en los patrones de conexión de las parejas IP-puerto de las aplicaciones P2P. Básicamente, cuando un nodo P2P inicia una conexión con un nodo, el puerto de destino será el puerto definido por el usuario de dicho nodo para atender las peticiones de la aplicación P2P.

Xu et al. [48] proponen un método para identificar tráfico P2P basándose en el comportamiento de la transferencia de datos de las aplicaciones P2P. Los autores aseguran que los datos descargados por un nodo de la red en las aplicaciones P2P serán subidos a otro nodo de la red con posterioridad. Así, dividen los datos descargados y subidos por los nodos en bloques de datos e intentan detectar aquellos flujos que comparten bloques de datos; éstos serán identificados como flujos P2P.

Otros trabajos se centran en la detección de actividades de red maliciosas como pueden ser: *spam*, ataques DoS, ataques DDoS, escaneo de puertos, etc. En esta línea se encuentra el trabajo de Soule et al. [49], que compara los resultados obtenidos utilizando cuatro métodos de clasificación diferentes. Otras aproximaciones para detectar anomalías de tráfico que utilizan *wavelets* son las expuestas en [50] y [51]. Lakhina et al. [52] utiliza la entropía como herramienta para resumir la información de los flujos de tráfico. En este trabajo se muestra que el análisis de las distribuciones

de características permite detectar un gran número de anomalías y clasificarlas automáticamente sin necesidad de supervisión.

2.4. Conclusiones

En este capítulo se pone de manifiesto la importancia de la clasificación de tráfico de red en la actualidad y, de ahí, la relevancia de este tema en la investigación.

Se han presentado los conceptos fundamentales relacionados con la clasificación de tráfico de red y que serán necesarios para la comprensión de los futuros capítulos.

La parte principal de este tema es la revisión de la bibliografía de investigación relativa a la clasificación de tráfico de red. Ésta se ha llevado a cabo poniendo un énfasis especial en la clasificación de tráfico P2P, pues éste es uno de los objetivos fundamentales de la presente tesis doctoral.

La revisión bibliográfica se ha agrupado según la fuente de información utilizada en: métodos de clasificación basados en puertos, basados en paquetes y basados en flujos. Al respecto de ellos:

- El primer grupo ha quedado sin uso por la propia evolución de las aplicaciones de red que ahora seleccionan puertos dinámicamente en su provisión.
- Los métodos de clasificación basados en paquetes son los más utilizados. En muchas ocasiones no pueden ser utilizados bien porque el contenido de los paquetes viaja encriptado o bien porque van contra la legalidad en un gran número de países del mundo por atentar contra la privacidad. Existen multitud de aproximaciones que se basan en la información de las cabeceras para eludir este inconveniente.
- Los métodos de clasificación de tráfico basados en flujos son más escalables pero suelen alcanzar tasas de detección menores que los basados en el contenido de los paquetes.

Los conceptos presentados en este capítulo constituyen la base de estudio sobre la cual se sustentan los sistemas de detección propuestos en los dos capítulos siguientes.

Capítulo 3

Identificación de tráfico eDonkey basada en heurísticas

El uso de aplicaciones basadas en redes P2P ha experimentado un incremento exponencial en la última década, lo que ha provocado que el volumen de tráfico generado por ellas llegue a suponer un gran porcentaje de todo el ancho de banda de Internet. Por este motivo, el interés de los proveedores de servicio de Internet por clasificar este tipo de tráfico ha aumentado también de forma considerable.

En este contexto, los protocolos más agresivos en lo que a consumo de ancho de banda se refiere son los utilizados para compartir recursos mediante P2P. Motivado por ello, y en el marco general que al respecto de la clasificación de tráfico de red ya se ha establecido en el capítulo anterior, en el presente capítulo se describen dos algoritmos de identificación o detección para el protocolo eDonkey (utilizado para compartir recursos mediante P2P). El primero de ellos, tiene como objeto de detección los flujos del protocolo y está basado en la hipótesis de que son flujos del protocolo eDonkey aquellos en los que el cliente que inicia la conexión envía sustancialmente más información de la que recibe. El segundo algoritmo ha sido desarrollado para detectar nodos generadores de tráfico eDonkey, y se basa en la hipótesis de que son nodos generadores de tráfico eDonkey aquellos cuya tasa de subida es constante y se comunican con múltiples direcciones IP.

Ambos algoritmos de detección han sido probados en tres conjuntos de trazas de tráfico de red real. Como resultado, se ha verificado la validez de las hipótesis en las que se basan los algoritmos de detección. Adicionalmente, los experimentos muestran que, pese a la sencillez de los algoritmos propuestos, éstos presentan una elevada tasa de aciertos y una baja tasa de falsos positivos.

El resto del capítulo ha sido estructurado en las siguientes cuatro secciones. En la Sección 3.1 se motiva la necesidad de diseñar técnicas de identificación de protocolos P2P y en concreto del protocolo eDonkey. En la Sección 3.2 se presentan conceptos generales de las redes P2P, junto a una descripción del funcionamiento del protocolo eDonkey. Los algoritmos de detección del protocolo eDonkey propuestos se detallan en la Sección 3.3. Por su parte, en la Sección 3.4 se presentan y discuten los resultados experimentales obtenidos en relación a los algoritmos de detección propuestos, junto a una breve descripción del entorno de experimentación considerado. Finalmente, en la Sección 3.5 se exponen las principales conclusiones extraídas tras la realización de este trabajo.

3.1. Motivación

El desarrollo y uso de aplicaciones basadas en redes P2P ha experimentado un crecimiento exponencial en los últimos años. Encontramos en la actualidad múltiples ejemplos de ellas, como pueden ser: eMule o uTorrent, como aplicaciones de compartición de ficheros, Skype, como aplicación de voz sobre IP (VoIP), Spotify, para la compartición de flujos de audio, y P2PTV, para la compartición de flujos de vídeo en tiempo real.

El tráfico generado por las aplicaciones P2P supone un enorme consumo del ancho de banda de la red. De hecho, en un estudio de 2008/2009 [2], Schulze et al. aseguran que el volumen de tráfico generado por aplicaciones P2P supone, dependiendo de la región, entre un 43 % y un 70 % de todo el ancho de banda de red. Este consumo de ancho de banda conlleva la aparición de retardos en las comunicaciones y, en suma, una disminución en la calidad de los servicios proporcionados.

Por ello, la capacidad de clasificar el tráfico P2P que consume un alto porcentaje de ancho de banda es una tarea de altísimo valor para los ISPs. Éstos no pueden asumir el elevado consumo de los servicios basados en redes P2P y se ven forzados a incrementar las operaciones de mantenimiento para controlarlo [53]. La disponibilidad de mecanismos eficientes y eficaces de clasificación de tráfico permitiría a los ISPs definir políticas de gestión de tráfico con las que ofrecer clases de servicio diferenciadas y aplicar a cada una de ellas una conformación del tráfico que dependa, por ejemplo, de las aplicaciones utilizadas por el usuario.

Un ejemplo claro de la importancia de esta clasificación se deriva del uso masivo de aplicaciones de compartición de recursos basadas en redes P2P como son eMule, aMule, uTorrent, etc. Dado el elevado consumo del ancho de banda asociado a estas aplicaciones, es común que el administrador de una red corporativa quiera controlar o incluso bloquear los flujos de tráfico de las mismas para que el ancho de banda de la empresa sea utilizado solamente con fines laborales. Para esta tarea es

necesario disponer de métodos de clasificación de tráfico que proporcionen resultados adecuados tanto desde el punto de vista de las tasas de detección (TPR y FPR) como desde el consumo de recursos y tiempo que ello implica.

El objetivo del presente capítulo es presentar dos aproximaciones diferentes para detectar uno de los protocolos para compartición de recursos mediante P2P más utilizados, el protocolo eDonkey. Éste es utilizado fundamentalmente por las aplicaciones eMule y aMule.

Para este fin, se presentan dos algoritmos para la detección de tráfico eDonkey, que son el resultado de un estudio detallado del comportamiento de este protocolo: (*i*) *detección de nodos* basada en la tasa de subida, URND, y (*ii*) *detección de flujos* basada en la inversión del sentido de la descarga, WCFD.

La primera, URND, utiliza la información de la tasa de subida de los nodos y se fundamenta principalmente en dos hipótesis: (*i*) los usuarios limitan la tasa de subida de las aplicaciones que utilizan eDonkey, y (*ii*) la tasa de subida tiene un comportamiento pseudo-constante a lo largo del tiempo, con ligeras variaciones alrededor del límite establecido por el usuario. De esta forma, si la tasa de subida de un nodo tiene un comportamiento constante a lo largo del tiempo alrededor de un mismo valor, este nodo será detectado como nodo generador de tráfico eDonkey.

Por su parte, WCFD se sustenta en la inversión lógica del sentido de la descarga, lo que es una característica particular de las aplicaciones de compartición de archivos que utilizan el protocolo eDonkey para comunicarse. En estas aplicaciones, el nodo que inicia la conexión es el nodo que envía la mayor parte de la información. Esto es radicalmente opuesto a lo que sucede habitualmente en el paradigma cliente-servidor, donde es el cliente quien inicia la conexión y el servidor, que la recibe, quien envía la mayor parte de información. Por tanto, si el sentido del envío de la información en un flujo es el contrario al común, es decir, la información viaja del nodo que inicia la conexión al que la recibe, este flujo será identificado como perteneciente al protocolo eDonkey.

3.2. Conceptos generales del protocolo eDonkey

Para comprender el funcionamiento de la detección propuesta en este capítulo es fundamental conocer los conceptos básicos del protocolo a detectar. A continuación se indican brevemente los conceptos más relevantes para el presente trabajo relativos a las redes P2P y más concretamente al protocolo eDonkey.

Las redes P2P se pueden dividir según su arquitectura en tres tipos: centralizadas, distribuidas e híbridas. En las redes *centralizadas* existe un servidor (denominado supernodo) encargado de indexar los recursos de la red, asociando los recursos a los

clientes que los comparten. En el caso de las redes *distribuidas* no se requiere ninguna gestión centralizada y los nodos son los encargados de almacenar la distribución de los contenidos en la red. Por último, las redes *híbridas* son una combinación de las dos anteriores. En éstas los nodos de la red pueden ser clientes y supernodos. Los clientes forman una red distribuida en torno a los supernodos, los cuales realizan las tareas de los servidores en las redes centralizadas.

El protocolo eDonkey se diseñó para la comunicación de nodos en una red P2P híbrida formada por supernodos (servidores de ahora en adelante) y clientes. Por un lado, los servidores dan acceso a la red, se encargan de manejar la distribución de la información en estructuras similares a diccionarios, que almacenan la relación entre los recursos y los nodos que los comparten. Por otro lado, los clientes son los únicos nodos que comparten datos, y son los encargados de almacenar físicamente una copia o parte de una copia de los recursos de la red.

3.2.1. Comunicaciones cliente-servidor en eDonkey

Para poder acceder a los recursos de la red, los clientes deben conectarse previamente a un servidor eDonkey. Esta conexión puede dividirse en dos fases (Figura 3.1): *(i)* solicitud de conexión por parte del cliente, y *(ii)* reto por parte del servidor.

1. *Solicitud de conexión*

   El cliente inicia una conexión TCP con el servidor y, posteriormente, envía el mensaje de registro del protocolo eDonkey, mensaje de tipo `login`. En este mensaje el cliente le indica al servidor el puerto de escucha desde el que puede recibir conexiones de otros clientes de la red eDonkey.

2. *Reto*

   El servidor responde al cliente con la segunda parte de la conexión, el reto. Para esto se realiza una segunda conexión TCP, en este caso iniciada por el servidor y cuyo destino es el puerto de escucha del cliente. Este reto determina si el cliente es capaz o no de recibir conexiones de otros clientes de la red.

El proceso de conexión finaliza con el envío, por parte del servidor, de un identificador único, ID, para el cliente que realizó la conexión. En caso de que el reto haya sido superado con éxito el cliente es identificado con un *ID alto*, en caso contrario con un *ID bajo*. Así, un ID bajo puede interpretarse como que el cliente no es capaz de aceptar conexiones de otros clientes de la red, mientras que un cliente con ID alto sí es capaz de recibirlas. La causa más común por la cual un cliente puede ser incapaz de recibir conexiones de otros nodos de la red eDonkey se debe a que la

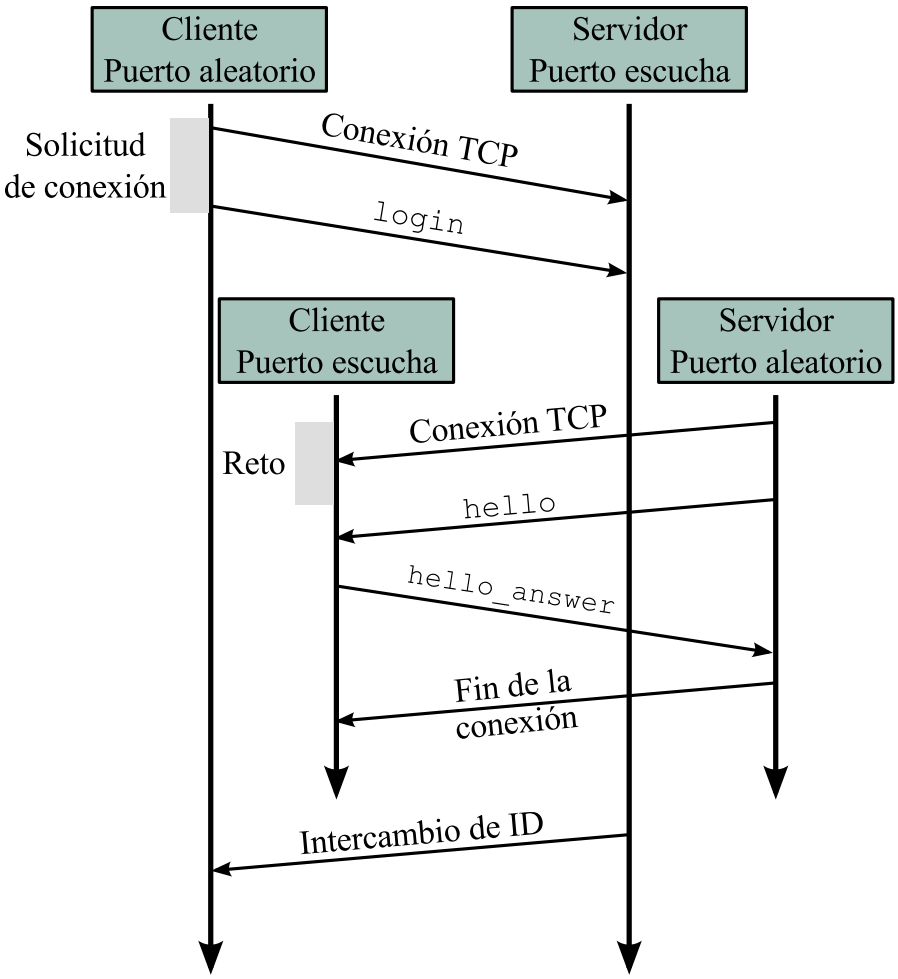


Figura 3.1: Proceso de conexión cliente-servidor en el protocolo eDonkey.

configuración del *router* de salida no permite las conexiones al puerto de escucha definido en la aplicación eDonkey.

Tras una conexión exitosa, tanto servidor como cliente intercambian información acerca de su estado actual. Por una parte, el cliente comienza enviando la lista de recursos que comparte y, por otra, el servidor envía su versión software y una lista con los servidores eDonkey que conoce.

Una vez que el cliente tiene acceso a un servidor eDonkey puede realizar búsquedas utilizando palabras clave. El servidor responderá a estas búsquedas con una lista de recursos que contienen la palabra o palabras clave. Posteriormente, el cliente podría descargar uno o más recursos de la lista obtenida por medio del envío de un mensaje al servidor, en el que solicita información del recurso concreto en el que está interesado. Para finalizar, el servidor responderá a esta solicitud con una lista de clientes que posean el recurso buscado.

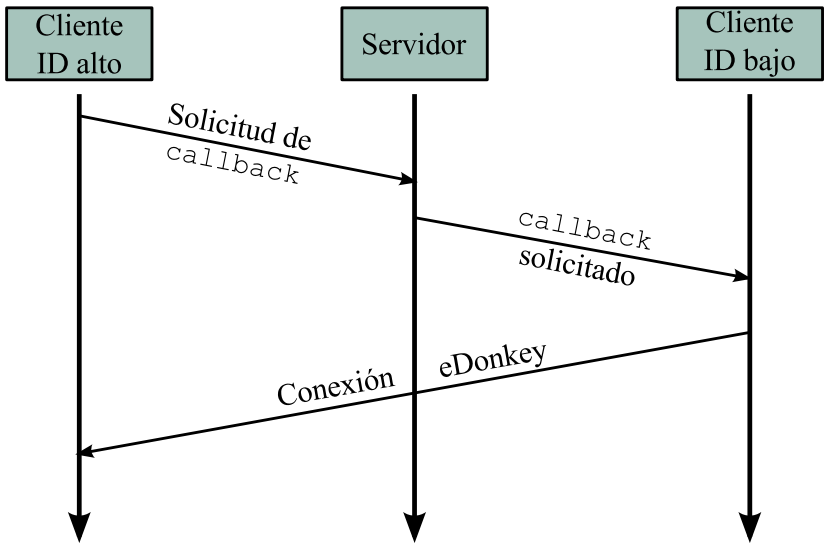


Figura 3.2: Conexión entre un cliente con un ID alto y uno con un ID bajo.

3.2.2. Comunicaciones cliente-cliente en eDonkey

Para descargar el recurso o recursos por los que el cliente ha preguntado al servidor, es necesario realizar previamente una conexión entre los clientes origen (el que inicia la solicitud) y destino (el que posee el recurso a descargar). El proceso normal de establecimiento una conexión cuando el cliente destino tiene un ID alto consta de los siguientes pasos:

1. Un cliente inicia una conexión TCP con la que se establece el canal para iniciar la comunicación eDonkey.
2. Se intercambian los mensajes del protocolo eDonkey: `hello` y `helloanswer`. En estos mensajes viaja información acerca del nodo que los envía, fundamentalmente el identificador de usuario y el puerto TCP configurado para recibir conexiones eDonkey.

Sin embargo, cuando el cliente destino posee un ID bajo, el procedimiento es diferente. Como se ha comentado, un cliente con un ID bajo no es capaz de aceptar conexiones, por lo que esta conexión resultaría imposible mediante el mecanismo convencional. Para solventar este problema, el protocolo eDonkey dispone del mensaje de tipo `callback`. Como puede verse en la Figura 3.2, el proceso de conexión entre un cliente origen con un ID alto y otro destino con un ID bajo aprovecha la conexión previamente establecida entre el cliente con ID bajo y el servidor. Así, el cliente origen envía un mensaje de tipo `callback` al servidor indicando su intención de conectarse con el destino con ID bajo. El servidor, en respuesta, envía otro mensaje de tipo `callback` al cliente con un ID bajo, a través de la conexión establecida

previamente con él (que fue iniciada por el propio cliente destino). Por último, el cliente destino, al recibir el mensaje del servidor, inicia una conexión con el cliente origen que sí es capaz de recibirlas al poseer un ID alto.

Por medio del uso de mensajes de tipo `callback` no es posible establecer la conexión entre dos clientes con IDs bajos, ya que ninguno de los dos es capaz de aceptar conexiones. En este caso, la conexión entre ellos no está soportada en el protocolo eDonkey.

El objetivo fundamental del establecimiento de una conexión eDonkey entre dos clientes es compartir un recurso concreto. El proceso más común en la compartición de un recurso mediante eDonkey se describe en lo que sigue (ver Figura 3.3).

- El cliente A (cliente origen) envía un mensaje solicitando el recurso buscado.
- El cliente B (cliente destino) responde a esta solicitud indicando que posee el recurso que A solicita.
- Posteriormente, el cliente B indica la posición que ocupa la solicitud del cliente A en su cola de peticiones a servir y cierra la conexión.
- Transcurrido un tiempo que depende de la ocupación de la cola de peticiones a servir del cliente B, cuando la petición del cliente A alcanza una posición en la misma que le permite ser servida, el cliente B inicia una conexión con el cliente A y le envía un mensaje indicando que acepta su solicitud de descarga.
- Finalmente, el cliente A solicita las partes concretas del recurso buscado para que el cliente B inicie su envío.

Tal y como se detalla en la Sección 3.3, el algoritmo de detección de patrones de generación de tráfico eDonkey se basa en este proceso.

Existe otra posibilidad, aunque es bastante infrecuente, y es que la solicitud del cliente A sea aceptada sin necesidad de cerrar la conexión iniciada por éste. Esto se puede deber a que la cola de servicio del cliente B no esté completamente llena, o a que, por algún motivo, B decida servir las peticiones del cliente A con una mayor prioridad.

3.3. Algoritmos de detección

Como ya se ha indicado con anterioridad, se han diseñado dos algoritmos de detección: (*i*) detección de nodos generadores de tráfico eDonkey basada en la tasa de subida, URND, y (*ii*) detección de flujos eDonkey basada en la inversión del sentido

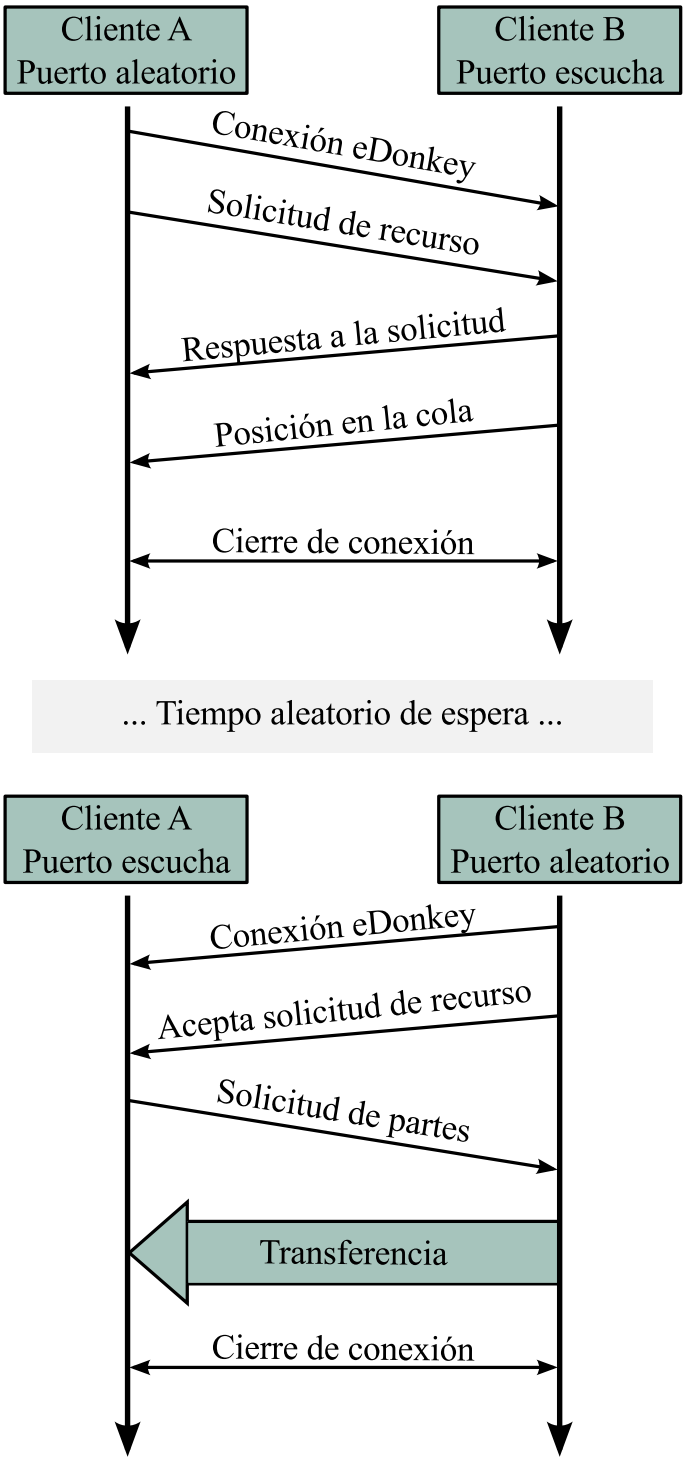


Figura 3.3: Entrada en la cola de peticiones de un cliente e inicio de la descarga de un recurso.

de la descarga, WCFD. Mediante URND se persigue determinar qué dispositivos generan en un momento determinado tráfico del protocolo, lo cual es de interés para los ISPs. Por otra parte, la detección de flujos usando WCFD pretende determinar qué flujos concretos pertenecen al protocolo eDonkey y puede resultar de gran interés de cara, por ejemplo, a la caracterización de los mismos para un filtrado más específico.

3.3.1. Detección de flujos eDonkey: WCFD

El primer algoritmo de detección tiene como objetivo la identificación de los flujos pertenecientes al protocolo eDonkey. Un flujo, como se vio en el capítulo anterior, es considerado de acuerdo con [46] como el tráfico identificado por la tupla `<IP-origen,puerto-origen,IP-destino,puerto-destino,protocolo>`, donde origen y destino son intercambiables para permitir tráfico bidireccional.

El algoritmo de identificación aquí presentado se basa en la hipótesis de que, a diferencia de lo que ocurre en la mayoría de las conexiones en el modelo cliente-servidor, el cliente destino en eDonkey, debido a que es el que debe iniciar la conexión hacia el cliente origen que le solicitó un recurso, será el que envíe la mayor parte de la información de la comunicación.

Al contrario de lo que sucede en la transferencia de recursos por medio del protocolo eDonkey, en la mayoría de las aplicaciones cliente-servidor es el servidor el encargado de enviar la mayor parte de la información tras una conexión iniciada por el cliente. Por este motivo, se propone la hipótesis de *inversión del sentido de la descarga* para la detección de flujos eDonkey:

Hipótesis 1: Inversión del sentido de la descarga
Son flujos del protocolo eDonkey aquellos en los que el cliente que inicia la conexión envía sustancialmente más información de la que recibe.

Es importante aclarar que esta hipótesis sólo se cumple para el caso de flujos del protocolo eDonkey utilizados en la transferencia de recursos y no para el caso de flujos de señalización. Por ejemplo, la primera conexión en el proceso de compartición de un recurso es un flujo de señalización, ya que sólo se utiliza para indicar la intención de iniciar la compartición de un recurso. En este caso no se cumpliría la hipótesis de detección antes señalada puesto que tanto el cliente A como el cliente B pueden enviar más información que el extremo opuesto de la comunicación.

Basado en la hipótesis de *inversión del sentido de la descarga*, se ha desarrollado el algoritmo de detección de flujos de transferencia de recursos del protocolo eDonkey,

Algoritmo 1 Detección de flujos (WCFD)

1: $flujos\_eD \Leftarrow \varnothing$
2: **para** $i = 0$ mientras $i < num\_flujos$ **hacer**
3: **si** $bytes\_env[i] > bytes\_rec[i] + \theta_B$ **entonces**
4: $flujos\_eD \Leftarrow flujos[i]$
5: **fin si**
6: $i \Leftarrow i + 1$
7: **fin para**
8: **devolver** $flujos\_eD$

descrito en el Algoritmo 1. Este algoritmo de detección ha sido diseñado para ejecutarse en modo *offline*, es decir, en capturas de tráfico almacenadas en el disco en lugar de en tiempo real ya que es necesario disponer de los flujos de tráfico completos para evaluar si se ha enviado más tráfico del que se ha recibido.

El proceso seguido por este algoritmo se resume en lo que sigue:

1. Se recorren uno a uno los flujos que conforman la captura y se extraen aquellos cuyo número de bytes enviados desde el cliente que inicia la conexión al que la recibe es mayor que la información recibida en sentido contrario más un umbral, θ_B.

2. Este umbral se debe determinar experimentalmente en base a un estudio de la distribución de tamaños de los flujos de transferencia de recursos del protocolo eDonkey.

3. Finalmente, se obtienen los flujos que el algoritmo detecta como flujos de compartición de recursos del protocolo eDonkey, $flujos\_eD$.

Como se verá en la Sección 3.4 la elección del umbral utilizado en este algoritmo de detección no resulta crítica, ya que la diferencia de tamaños en los flujos de compartición de archivos del protocolo eDonkey es muy elevada.

3.3.2. Detección de nodos eDonkey: URND

El objetivo del segundo algoritmo propuesto es detectar los nodos generadores de tráfico eDonkey. Esta detección permitiría aplicar políticas de conformación de tráfico a los nodos detectados. El algoritmo propuesto sigue una alternativa distinta al de detección de los flujos pertenecientes al protocolo eDonkey, aunque podría ser complementario a aquélla.

El comportamiento usual de los usuarios de las aplicaciones P2P que utilizan el protocolo eDonkey como eMule o aMule consiste en limitar la tasa de subida de éstas para evitar la saturación de sus conexiones a Internet. Ello se debe a que las aplicaciones de compartición de recursos mediante P2P, como ya se ha comentado, consumen un gran porcentaje del ancho de banda de red disponible, lo que provoca una disminución considerable de la velocidad de la navegación web y, por tanto, de la calidad de servicio experimentada por los usuarios.

Tomando como base lo anterior, la hipótesis que sustenta URND es que la limitación de la tasa de subida establecida por el usuario en la aplicación eDonkey implica, la mayor parte del tiempo, una tasa de subida constante similar al límite configurado. Es posible que un nodo presente un tasa de subida constante por ejemplo al subir un archivo a un servidor de Internet. Sin embargo, algo característico de los nodos eDonkey es que la tasa de subida constante es el resultado de comunicaciones IP con muchos nodos diferentes. Así pues, esta tasa constante representa un comportamiento muy característico que puede permitir diferenciar un nodo generador de tráfico eDonkey de otro nodo. En conclusión, la hipótesis de detección queda como sigue:

Hipótesis 2: Tasa de subida constante
Son nodos generadores de tráfico eDonkey aquellos cuya tasa de subida es constante y se comunican con múltiples direcciones IP.

El reto principal de este algoritmo de detección es cómo determinar qué es un nivel constante de la tasa de subida. La respuesta a esta cuestión no es trivial y para abordarla se toma como referencia el trabajo [54], en el que los autores utilizan la divergencia de KL (*Kullback Leibler*) para detectar la actividad de voz en señales de audio. Esta detección consiste en determinar el instante temporal en el que la señal de audio evaluada pasa de ser únicamente ruido a contener también voz. Esta transición viene caracterizada por un cambio en la media y la varianza de la señal de audio, que se detecta gracias a la divergencia de KL. La detección de un nivel constante, fundamental en el algoritmo propuesto, representa el caso contrario: se debe detectar una ausencia de cambios significativos en la tasa de subida (tasa constante) para determinar que un nodo está generando tráfico eDonkey.

La divergencia de KL puede ser interpretada como un indicador de la similitud entre dos funciones de distribución. Su expresión matemática es como sigue:

$$H(p_I \| p_D) = \int p_I(x) \log(\frac{p_I(x)}{p_D(x)}) dx \tag{3.1}$$

donde p_I y p_D son dos funciones de distribución que pretenden ser comparadas por medio de la divergencia de KL.

Dependiendo de las funciones de distribución $p_I(x)$ y $p_D(x)$ la integral de la divergencia de KL puede resultar muy compleja de resolver, e incluso no tener solución. Sin embargo, para el caso en el que $p_I(x)$ y $p_D(x)$ sean ambas distribuciones gaussianas, la divergencia de KL se simplifica:

$$H(p_I \| p_D) = \frac{1}{2}[log(\frac{\sigma_D^2}{\sigma_I^2}) - 1 + \frac{\sigma_I^2}{\sigma_D^2} + \frac{(\mu_I - \mu_D)^2}{\sigma_D^2}] \tag{3.2}$$

donde σ_I y σ_D representan las desviaciones típicas de p_I y p_D, y μ_I y μ_D sus medias.

La divergencia de KL expresada en la Ecuación (3.2) no es simétrica, lo que quiere decir que $H(p_I \| p_D)$ puede ser diferente de $H(p_D \| p_I)$. Esto implica que la desviación de dos funciones de distribución en un sentido podría presentar una divergencia distinta que la misma desviación en el sentido contrario. Esto no es deseable, así que en la Ecuación (3.3) se muestra la divergencia de KL de dos distribuciones gaussianas p_I y p_D en su forma simétrica:

$$\rho_{I,D} = \frac{1}{2}\left[\frac{\sigma_I^2}{\sigma_D^2} + \frac{\sigma_D^2}{\sigma_I^2} - 2 + (\mu_I - \mu_D)^2(\frac{1}{\sigma_I^2} + \frac{1}{\sigma_D^2})\right] \tag{3.3}$$

En el caso de [54] se utilizan los máximos de la divergencia de KL para encontrar los instantes en los que la señal de audio pasa de ser únicamente ruido a contener voz, o viceversa. Esto se debe a que en este trabajo la señal de audio es recorrida por dos ventanas consecutivas, v_I y v_D, y los datos contenidos en ellas son representados por dos distribuciones gaussianas a las que se les calcula la divergencia de KL. Así, el máximo de la divergencia mostrará el punto en el que la señal de audio presenta un cambio significativo en esas dos distribuciones, esto es, cuando la señal pasa de contener sólo ruido a contener también señal de voz.

El presente algoritmo se basa en el trabajo anterior, pero en este caso se pretende detectar una tasa de subida constante, es decir, que las distribuciones gaussianas que representan los datos de las ventanas consecutivas sean lo más similares posible, lo que implica que la divergencia de KL sea cercana a cero. En el algoritmo propuesto esto se describirá por medio de un umbral que no debe superar la divergencia de KL, θ_{KL}. Este umbral se determinará de forma experimental como se verá en la Sección 3.4.

En resumen, el algoritmo de detección derivado de la hipótesis de *tasa de subida constante* puede ser descrito como sigue (véase Algoritmo 2):

1. En primer lugar, los valores de la tasa de subida de un nodo son calculados en intervalos de δ segundos.

Algoritmo 2 Detección de nodos (URND)

para $nodo = 0$ mientras $nodo < num\_nodos$ **hacer**
  para $i = 0$ mientras $i < len(tasa\_sub_{nodo})$ **hacer**
    $tasa\_filt_{nodo} \Leftarrow filt\_mediana(tasa\_sub_{nodo}, N)$
    $v_I \Leftarrow tasa\_filt_{nodo}[i : i + N - 1]$
    $v_D \Leftarrow tasa\_filt_{nodo}[i + N : i + 2N - 1]$
    $\rho_{I,D}[i] \Leftarrow \frac{1}{2}\left[\frac{\sigma_I^2}{\sigma_D^2} + \frac{\sigma_D^2}{\sigma_I^2} - 2 + (\mu_I - \mu_D)^2(\frac{1}{\sigma_I^2} + \frac{1}{\sigma_D^2})\right]$
    si $\rho_{I,D}[i] < \theta_{KL}$ AND $\mu_{I,D}$ distinto a 0 **entonces**
      devolver Sí eDonkey
    si no
      devolver No eDonkey
    fin si
    $i \Leftarrow i + 1$
  fin para
  $nodo \Leftarrow nodo + 1$
fin para

2. El elevado *churn* de las redes P2P deriva en que la tasa de subida fluctúa bastante alrededor del valor constante. Para minimizar esta fluctuación los valores del paso anterior son filtrados paso baja mediante un filtro de mediana [55] de tamaño N. Este filtro toma N valores de tasa de subida (una ventana de tamaño N) y los ordena de menor a mayor, quedándose con el valor que ocupa el centro del intervalo.

3. La señal resultante del filtro de mediana anterior es recorrida por dos ventanas consecutivas (v_I y v_D), cada una de tamaño N (Figura 3.4). En cada ventana se computa de forma independiente la media y la varianza de los valores en ellas comprendidos, σ_I y μ_I, σ_D y μ_D. Se asume que los valores de cada ventana pueden representarse por distribuciones gaussianas de media y varianza σ_I y μ_I, σ_D y μ_D y se calcula la divergencia de KL simétrica (Ecuación (3.3)) de estas distribuciones gaussianas.

4. Una tasa de subida constante implica que las distribuciones de ambas ventanas son muy similares y, por consiguiente, la divergencia de KL será cercana a cero. Así, si el valor de la divergencia KL de las ventanas evaluadas es menor que el umbral θ_{KL}, la tasa de subida del nodo en ese intervalo es considerada constante. Si a esto se añade que la tasa de subida es distinta de cero, el nodo evaluado será clasificado como nodo generador de tráfico eDonkey en ese intervalo.

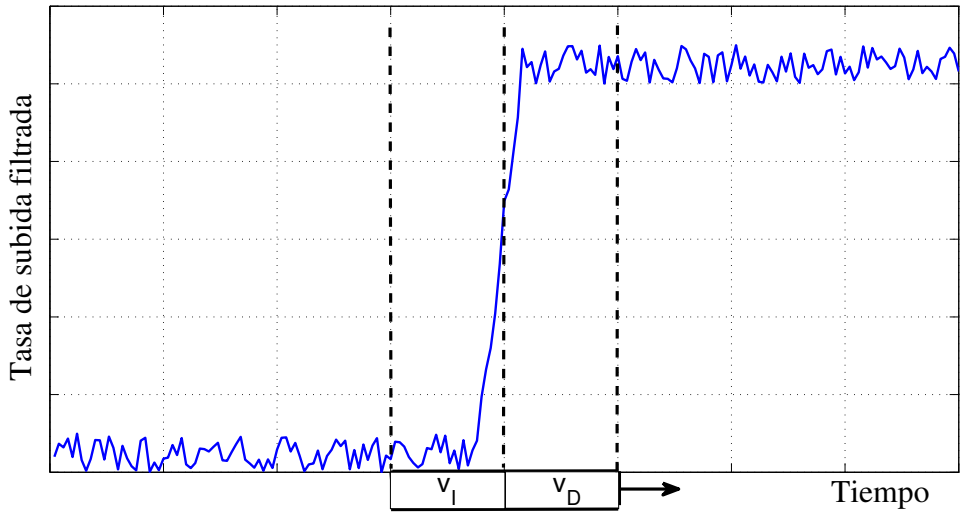


Figura 3.4: Avance con el tiempo de las ventanas de cálculo de la divergencia de Kullback-Leibler.

3.4. Resultados experimentales

La experimentación realizada para probar el buen comportamiento de los algoritmos de detección propuestos en este capítulo se ha centrado principalmente en dos objetivos. Por un lado, estudiar la validez de las hipótesis en las que se basan los algoritmos de detección planteados: la hipótesis de inversión del sentido de la descarga en WCFD y la hipótesis de tasa de subida constante en URND. Y por otro, analizar si estas hipótesis se cumplen únicamente para el protocolo eDonkey o cuando se aplican a otros protocolos diferentes de eDonkey presentan tasas de falsos positivos elevadas.

A continuación se describe, en primer lugar el entorno experimental utilizado y, posteriormente, se analizan separadamente los resultados obtenidos para los algoritmos de detección de flujos y de nodos.

3.4.1. Descripción del entorno experimental

Se han utilizado tres conjuntos de capturas de tráfico para realizar la experimentación relativa a los algoritmos de detección propuestos.

- *Trazas recopiladas en un entorno controlado (EC)*

  Un total de 5 usuarios aceptaron voluntariamente someterse a la monitoriza-

ción de su tráfico de red durante 72 horas ininterrumpidas. En este período compartieron, mediante el programa aMule en su versión 2.2.6, una carpeta de archivos con idéntico contenido. Todos ellos se conectaron al servidor de eDonkey *se-Master Server 1* y limitaron la tasa de subida de aMule a 30 KB/s.

Todos los usuarios utilizaron sus PCs y su conexión a Internet sin ningún tipo de restricción. En resumen, en media cada usuario generó alrededor de 19.000 conexiones del protocolo eDonkey y más de 7.000 de otros protocolos, entre los que cabe destacar DNS, HTTP, SSH y SMTP (*Simple Mail Transfer Protocol*).

- *Trazas de un servicio web (SW)*

  Este conjunto de trazas contiene el tráfico generado y recibido por un servidor web de una universidad europea durante 7 días. Es un servidor Apache versión 2.2.0 que recibe una media de 8.971 conexiones por día.

- *Trazas de un troncal de universidad (TU)*

  Estas trazas están formadas por todo el tráfico saliente del troncal de una universidad de Oriente Medio durante 48 horas. En este conjunto de trazas se encuentran alrededor de 73.000 direcciones IP, se han transmitido cerca de 300 millones de paquetes y los principales protocolos utilizados son: Bittorrent, HTTP, DNS, SSL (*Secure Sockets Layer*) y FTP. entre otros.

  Tras el análisis de toda la base de datos mediante una inspección de paquetes con la aplicación OpenDPI [16], no se ha detectado ningún paquete del protocolo eDonkey. Esto puede deberse a que las aplicaciones P2P de compartición de recursos utilizadas en esta universidad de Oriente Medio se basan en Bittorrent en lugar de eDonkey, ya que de este protocolo sí se han detectado paquetes.

Para la verificación de las hipótesis de detección, se utilizarán las trazas del entorno controlado, EC, gracias a las que se podrá extraer la tasa de aciertos TPR de los algoritmos propuestos. Por otra parte, para comprobar si estas hipótesis se cumplen también para otros protocolos se utilizarán las trazas del servidor web, SW, y del troncal de la universidad de oriente medio, TU, y así se hallarán las tasas de falsos positivos FPR asociadas a los algoritmos de detección.

3.4.2. Detección de flujos: WCFD

En primer lugar, es necesario un entrenamiento del algoritmo de detección para determinar el umbral óptimo de diferencia de bytes enviados frente a recibidos, θ_B. Para esto, se ha realizado un estudio del porcentaje de flujos detectados en un 10% de los tres conjuntos de trazas en función del valor de θ_B resultando la curva ROC mostrada en la Figura 3.5. Como puede verse en esta figura, los resultados de este

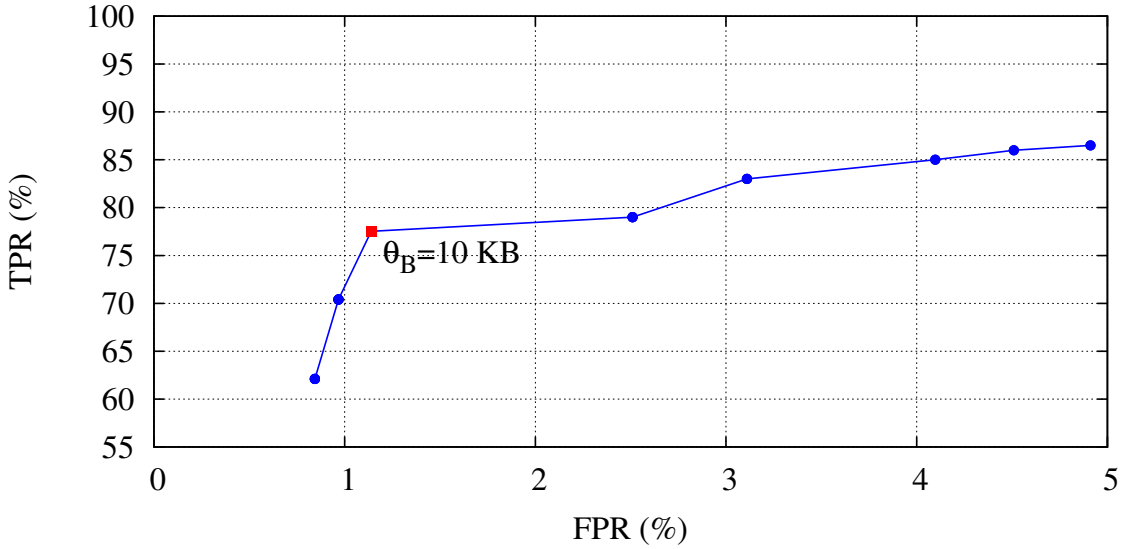


Figura 3.5: ROC del algoritmo WCFD variando el valor de θ_B.

análisis indican que existe un amplio rango para la elección de este umbral dentro del cual la eficacia de este algoritmo de detección es considerable. Concretamente, el valor de θ_B utilizado para la experimentación de detección de flujos es de 10 KB, lo que supone que la información enviada en un flujo por el cliente que inicia la conexión debe superar en más de 10 KB a la enviada por el que recibe la conexión para que este flujo sea detectado, por el algoritmo propuesto, como flujo del protocolo eDonkey. Este punto de operación implica un TPR de más del 75% y un FPR de menos del 2%.

Para evaluar la tasa de aciertos y de falsos negativos, en primer lugar se evalúan los resultados relativos al 90% de las trazas recopiladas en entorno controlado no utilizadas en el entrenamiento del algoritmo. Las trazas EC contienen 37.089 flujos de compartición de archivos eDonkey. De éstos, 28.016 han sido detectados correctamente, lo que supone una tasa de acierto, TPR, del 77,53%. En este conjunto de trazas no se ha producido ningún falso positivo.

El porcentaje de falsos negativos, es decir, de flujos de compartición de archivos de eDonkey que no han sido detectados, asciende a un 22,47%. Este porcentaje es considerable pero se debe a dos causas que hacen que los flujos no detectados queden fuera de la hipótesis de detección propuesta. Éstas son:

1. *ID bajo en alguno de los extremos*

   Los nodos con ID bajo no pueden aceptar conexiones de la red eDonkey, por lo que siempre inician la conexión, independientemente de si son ellos los que han de enviar la información en la comunicación. Esta situación queda, por

Tabla 3.1: FPR del algoritmo de detección de flujos sobre el conjunto de trazas TU.

	FPR	Flujos detectados	Flujos totales
BitTorrent	2,561 %	854	33.304
HTTP	1,691 %	50.795	3.003.161
FTP	1,423 %	35	2.460
SSL	1,244 %	2.808	225.685
IRC (*Internet Relay Chat*)	0,213 %	7	3.281
Oscar	0,079 %	2	2.528
DNS	0,001 %	8	1.508.413
Mail_POP	0,000 %	0	5.208
Todos	1,139 %	54.509	4.784.040

tanto, fuera de la hipótesis de inversión del sentido de la descarga en la que se basa el algoritmo de detección.

2. *Compartición sin cierre intermedio de conexión*

   Aunque es menos frecuente, es posible que la solicitud de la descarga de un recurso llegue a ser servida sin necesidad de cerrar la conexión inicial. Esto se puede producir bien porque la cola de servicio del cliente con el que se realiza la conexión no esté llena, o porque la prioridad del cliente que solicita la descarga sea muy alta (véase Sección 3.2). Si esto es así, la hipótesis de inversión del sentido de la descarga tampoco es válida.

Realizado el análisis de detección de tráfico exclusivo eDonkey, seguidamente abordamos la identificación sobre las bases de datos SW y TU, lo que implica que todo flujo detectado representará un falso positivo. Para ello, se ha evaluado el 90 % de estas trazas no utilizadas en el entrenamiento del algoritmo de detección. Primeramente se han etiquetado los protocolos a los que corresponden los diferentes flujos que conforman estas trazas. Para ello, se ha modificado la aplicación `OpenDPI` para transformarla en una aplicación de clasificación de flujos en lugar de clasificación de paquetes.

Los resultados de la detección se muestran en la Tabla 3.1. BitTorrent es el protocolo que mayor tasa de falsos positivos presenta (2,56 %) y esto se debe a que, al ser también un protocolo de compartición de recursos mediante P2P puede ser coherente en algunos casos con la hipótesis presentada. A diferencia de eDonkey, los flujos BitTorrent son bidireccionales, es decir, a través del mismo flujo ambos extremos de la comunicación pueden enviar partes de un recurso. Por este motivo, el algoritmo no detecta un porcentaje mayor de flujos BitTorrent. Por otro lado, es lógico que los flujos FTP sean detectados también como generadores de tráfico eDonkey, pues es normal que el cliente envíe por FTP más que el servidor (*p.ej.*, el caso de subida de ficheros al servidor).

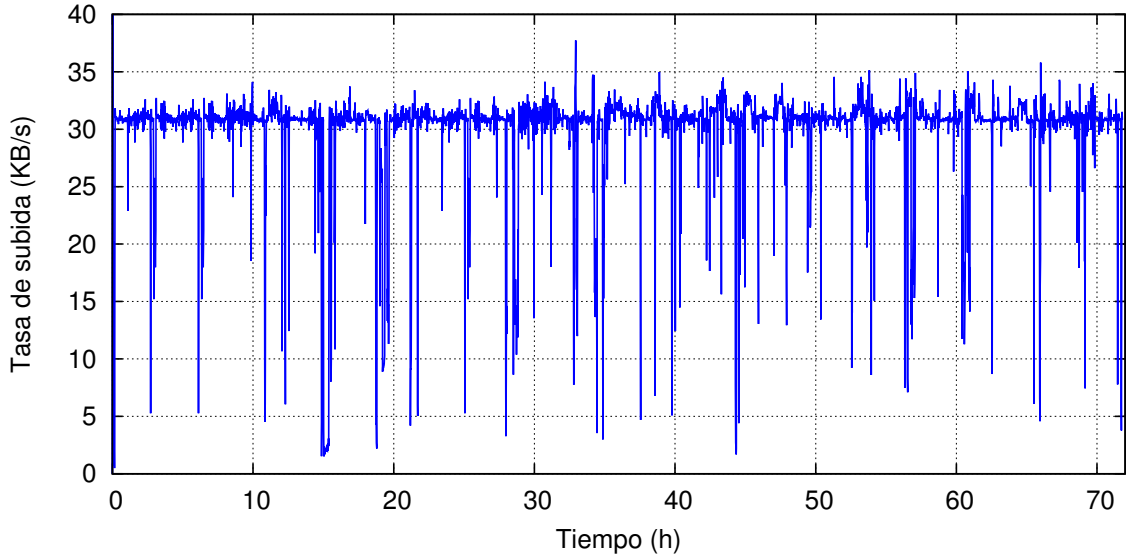


Figura 3.6: Tasa de subida de uno de los nodos monitorizados en la traza EC.

Es notable el resultado de detección asociado al protocolo HTTP ya que, aunque no fue detectado ningún flujo en las trazas SW, en éstas aparece un 1,69 % de falsos positivos. Tras estudiar en detalle estos falsos positivos se descubrió que son provocados principalmente por tres motivos:

1. Campos de *cookie* y URL extremadamente extensos en el mensaje HTTP `GET`, que provocan que el cliente envíe más información que el servidor.

2. Respuestas del servidor con código 304 (Recurso no modificado), que son muy reducidas en tamaño y, por tanto, menores que la petición.

3. Peticiones HTTP `POST`, en las que el envío de datos de formulario al servidor hacen que la petición pueda ser más grande que la respuesta.

3.4.3. Detección de nodos: URND

El algoritmo de detección de nodos propuesto y detallado en la Sección 3.3.2 se basa en la hipótesis de tasa de subida constante. Como puede verse en la Figura 3.6, en la que se representa la tasa de subida para uno de los nodos monitorizados, la tasa de subida de los nodos eDonkey se comporta de forma coherente a lo hipotetizado. Como era de esperar, durante las 72 horas de duración de la traza se puede apreciar claramente un comportamiento constante alrededor de 30 KB/s, que es el límite superior impuesto por el usuario en el experimento.

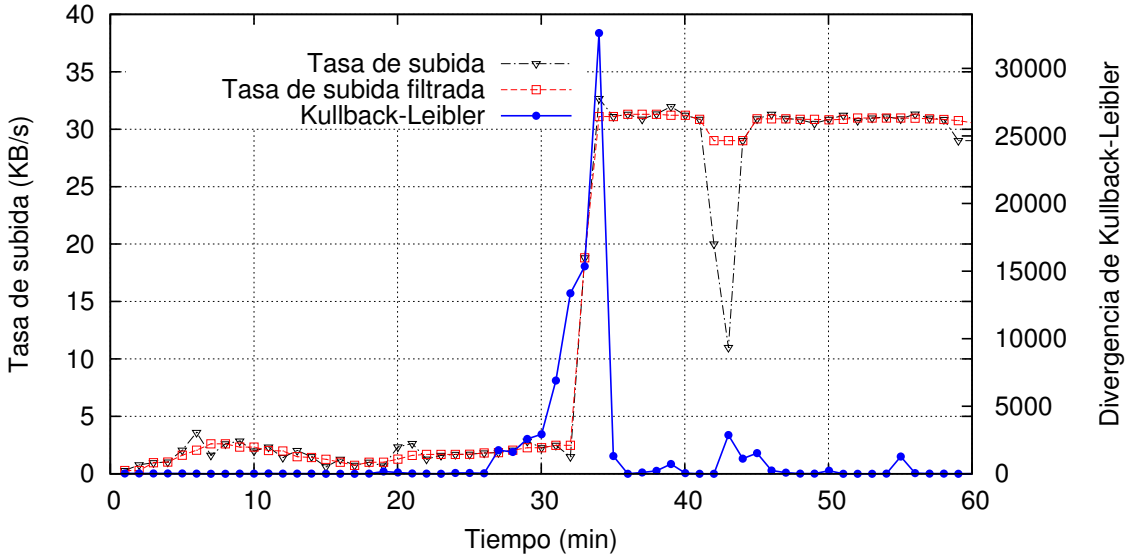


Figura 3.7: Evolución de la divergencia de Kullback-Leibler en el análisis de la primera hora de monitorización de un nodo (traza EC).

También se pueden apreciar en esta figura decaimientos de la tasa de subida. Éstos son de duración reducida y corresponden a instantes en los que se dejan de enviar datos a un nodo para iniciar la transferencia con otro. El *churn* de las redes P2P (tasa de entrada y salida de nodos a la red) es elevado y es la principal razón por la que se producen estos decaimientos.

Los instantes en los que se supera el límite establecido por el usuario son otra característica a resaltar de la Figura 3.6. Éstos corresponden a actividad de red adicional al tráfico eDonkey, como puede ser: navegación HTTP, subida de algún archivo mediante SSH, envío de correo, etc. En estas trazas, un 38,7 % de los flujos pertenecen a protocolos diferentes a eDonkey, entre los que destacan DNS, HTTP, SSH y SMTP.

En la Figura 3.7 se muestra la tasa de subida de uno de los usuarios de la traza EC en la primera hora del experimento. Durante los primeros 30 minutos apenas hay tasa de subida porque el servidor aún no ha dado a conocer a suficientes usuarios la existencia de este nuevo nodo en la red. A partir de esa primera media hora se aprecia un comportamiento constante de la tasa de subida alrededor de 30 KB/s. También se puede ver la reducción del efecto de los valores de tasa de subida que se desvían excesivamente de los esperados gracias a la aplicación del filtro de mediana.

Por último, en esa misma representación se superpone la divergencia de KL de los valores filtrados de la tasa de subida. La media del primer tramo de la divergencia de KL es cercana a cero y la varianza reducida. Esto se debe a que la tasa de subida en este intervalo es muy similar. En el segundo tramo se puede observar un incremento

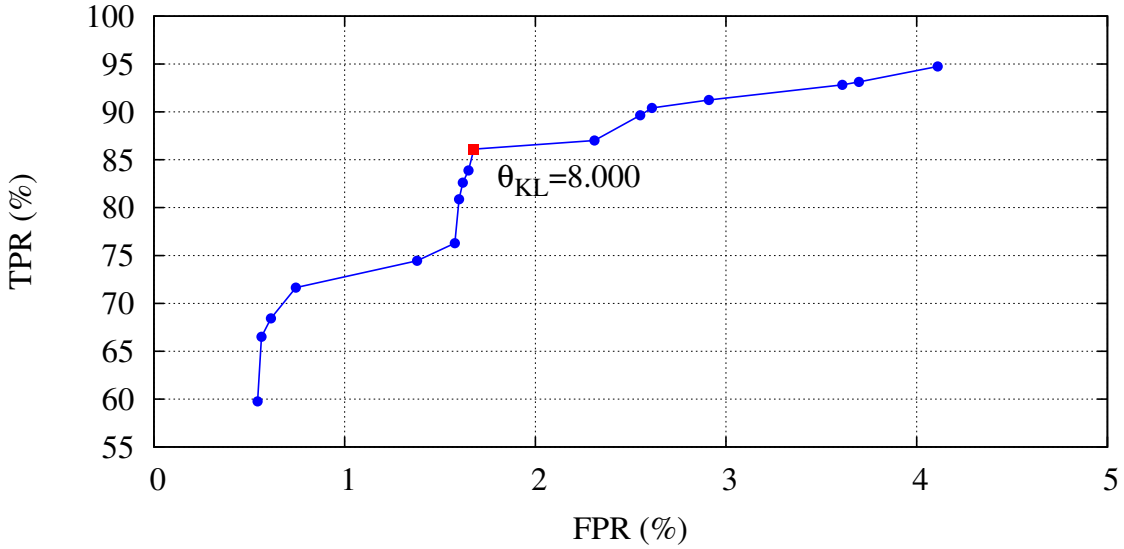


Figura 3.8: ROC del algoritmo URND variando el valor de θ_{KL}.

en la divergencia de KL, debido al abrupto cambio de media y varianza de la tasa de subida. Por último, la media y varianza de la divergencia vuelven a ser reducidas en el tercer tramo de la representación. Este último tramo se detecta como tráfico generado por un nodo de la red eDonkey al tener una divergencia KL cercana a cero y una media de la tasa de subida superior a cero.

Es importante realizar una primera fase de entrenamiento para escoger el valor del umbral, θ_{KL}, y el tamaño de la ventana, N, del filtro de mediana y del cálculo de la divergencia de KL. En primer lugar, se ha buscado un valor de θ_{KL} que maximice la detección de los nodos generadores de tráfico eDonkey y minimice la aparición de falsos positivos. Para ello, se ha ejecutado el algoritmo de detección sobre una parte (un 10 %) de los tres conjuntos de trazas (EC, SW y TU) con valores de θ_{KL} variando en el rango de 10^2 a 10^5. Los resultados de detección se muestran en la ROC de la Figura 3.8, en la que existe un amplio rango de valores para θ_{KL} que no presentan grandes variaciones en las tasas de detección y de falsos positivos. Este resultado permite a los autores asegurar que la elección de θ_{KL}, dentro de un rango amplio $[10^3, 10^4]$, no es crítica. Para los resultados de test expuestos a continuación y realizados sobre el resto de las trazas de tráfico (90 % no utilizado en el entrenamiento) se ha seleccionado un valor de divergencia de KL para θ_{KL} igual a 8.000 (con un TPR mayor de 85 % y un FPR menor de 2 %).

En segundo lugar, el tamaño de ventana, N, del filtrado de mediana y del cálculo de la divergencia de KL, se ha fijado a 5 minutos por ser ésta la mínima ráfaga de tráfico constante que se pretende detectar. Un tamaño de ventana mayor dificultaría enormemente la detección de ráfagas de este tamaño. Las ráfagas de un tamaño

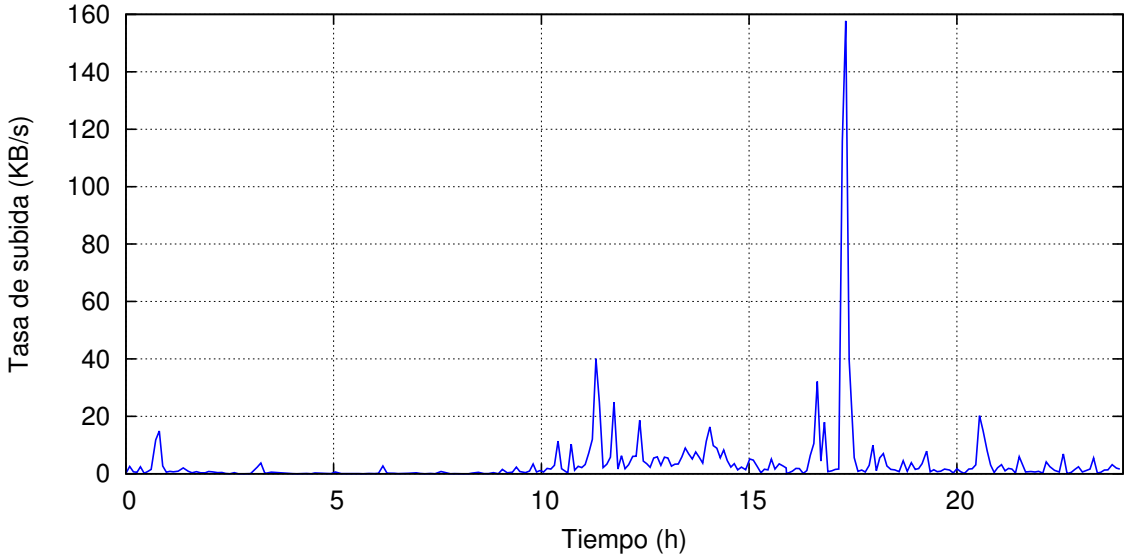


Figura 3.9: Tasa de subida del servidor HTTP (traza SW) durante 24 horas.

menor no son objeto de este trabajo ya que se asume que un nodo generador de tráfico eDonkey interesante por su elevado consumo de ancho de banda permanece conectado a la red un tiempo prolongado.

El porcentaje de tiempo durante el cual, según el algoritmo propuesto, los nodos han estado generando tráfico eDonkey ha sido de un 86,10 %. Los intervalos de la tasa de subida que no se han detectado como constantes se relacionan con los decaimientos provocados por el cambio de usuario con el que se comparten recursos (generalmente debidos al elevado *churn* propio de las redes P2P).

Por último, también se ha comprobado, como puede verse en la Figura 3.9, que la tasa de subida del servidor HTTP no parece tener un comportamiento constante con media superior a cero. Este mismo comportamiento, aparentemente aleatorio, se observa en los nodos monitorizados en la traza TU. La tasa de falsos positivos es de un 1,678 %, lo que implica que sólo un 1,678 % del tiempo total de monitorización en ambas trazas de tráfico se detectó alguna ráfaga de más de 5 minutos de duración de tasa de subida constante distinta de cero.

3.4.4. Comparación de los resultados de clasificación obtenidos

Los resultados de clasificación de ambos algoritmos presentan unos niveles de detección considerablemente buenos pero su principal ventaja es la sencillez de la aproximación propuesta, lo que permite una eficiencia computacional muy elevada. Por un lado, el algoritmo de detección de flujos, WCFD, alcanza una TPR de

77,53 % con una FPR de 1,139 %. Por otro, el algoritmo de detección de nodos, URND, presenta una TPR de 86,10 % con una FPR de 1,678 %.

Ambos algoritmos mejoran ampliamente los resultados de detección para protocolos de compartición de recursos del trabajo de Moore et al. [15] que obtenía una tasa de acierto de 36,45 %. La aproximación presentada por el algoritmo WCFD es igualmente válida para flujos ofuscados, a diferencia del trabajo de Sen et al. en [29], y también mejora ampliamente las tasas de falsos positivos de este último trabajo, que estaban en torno al 5 %. Por último, en el trabajo de Bernaille et al. [38] el sistema de detección propuesto alcanza una TPR de 84,2 % para el protocolo eDonkey que es similar a la que se obtiene con el algoritmo URND y un poco superior a la obtenida con WCFD, pero no indican en este trabajo su FPR.

3.5. Conclusiones

En el presente capítulo se aborda el problema de detectar el tráfico perteneciente al protocolo eDonkey sin inspeccionar el contenido de los paquetes. Para este fin se proponen dos algoritmos de detección. El primero tiene como objeto la detección de flujos y el segundo la detección de nodos eDonkey.

El algoritmo de detección de flujos se basa en la afirmación de que los flujos del protocolo eDonkey cumplen que el número de bytes enviados desde el cliente que inicia la conexión al que la recibe es sustancialmente mayor que en el sentido inverso.

El segundo algoritmo de detección afirma que son nodos generadores de tráfico eDonkey aquellos cuya tasa de subida es constante y se comunican con múltiples direcciones IP.

En base a la experimentación realizada con los tres conjuntos de trazas dispuestos a tal efecto se puede concluir que las hipótesis de detección propuestas se cumplen para el protocolo eDonkey, presentando una buena tasa de detección y un reducido número de falsos positivos en comparación con los trabajos de la literatura. La experimentación también ha permitido especificar que el algoritmo de detección de flujos es válido únicamente en flujos de compartición de archivos para nodos eDonkey con ID alto.

Capítulo 4

Clasificación de tráfico P2P mediante modelado de Markov

En el capítulo anterior se presentaron dos algoritmos de identificación de tráfico para un protocolo P2P concreto, basados en las características intrínsecas del mismo. Un claro inconveniente de estos algoritmos de detección es que sólo pueden identificar un protocolo concreto y, adicionalmente, son muy dependientes de la implementación específica de éste, pudiendo ser inservibles con un cambio leve del mismo.

Por esta razón, en el presente capítulo se pretende abordar el problema de la clasificación de tráfico P2P proponiendo un sistema de detección que no se centre en un único protocolo sino que se pueda adaptar fácilmente para detectar múltiples protocolos. Este sistema de detección se basa en el uso de modelos de Markov para clasificar tráfico de red, y trabaja a nivel de flujo considerando los paquetes de red como observaciones entrantes del modelo. Adicionalmente, es capaz de analizar tanto comunicaciones encriptadas como sin encriptar. Como caso de estudio, en el presente capítulo se aplica este esquema con el objetivo de identificar flujos del protocolo de compartición de archivos eDonkey, para así poder comparar los resultados con los obtenidos en el capítulo anterior.

Para validar el sistema de detección propuesto en este capítulo se presenta una evaluación del mismo con tráfico de red real. Los resultados experimentales obtenidos demuestran el buen comportamiento del sistema tanto en tasa de aciertos como de falsos positivos.

El resto del capítulo se organiza como sigue. En la Sección 4.1 se exponen con brevedad los principios fundamentales del modelado de Markov. Posteriormente, la parametrización y estructura del modelo de Markov utilizado para la clasificación de tráfico de red aquí propuesto se presenta en la Sección 4.2, concretando finalmente la configuración específica para el caso de estudio, el protocolo eDonkey. La evaluación del funcionamiento del sistema de detección para el caso de estudio se lleva a cabo analizando dos conjuntos de datos de tráfico en la Sección 4.3. Aquí, se obtienen resultados que señalan el buen funcionamiento de la metodología propuesta. Finalmente, las principales conclusiones de este capítulo se muestran en la Sección 4.4.

4.1. Fundamentos del modelado de Markov

En teoría de probabilidad, el modelado de Markov se refiere a un modelo que asume la propiedad de Markov para un proceso dado. Esto quiere decir que la probabilidad de estados futuros depende únicamente del estado actual.

La fortaleza del modelado de Markov reside en su capacidad para representar el comportamiento dinámico de un sistema. El modelado de Markov fue introducido por primera vez en 1954 [56] y ha sido aplicado a multitud de campos, desde reconocimiento del habla a medicina, pasando por sismología e ingeniería [57].

Los dos tipos de modelos de Markov más comúnmente utilizados son: cadenas de Markov y modelos ocultos de Markov (HMM). En el primer caso, cada estado del sistema es modelado a través de una variable aleatoria que cambia en función del tiempo. Un HMM, por su parte, es una cadena de Markov en la que el estado es parcialmente observable. Las observaciones están relacionadas con el estado pero no contienen información suficiente como para determinar con precisión el estado al que pertenecen. Los HMM son la técnica predominante en clasificación [58] y es en éstos en los que nos centraremos en el resto del capítulo.

4.1.1. Conceptos generales en HMM

Dada una cadena de Markov discreta con un conjunto de estados finitos $S = \{s_1, s_2, ..., s_N\}$, se define un HMM como la tupla $\lambda = (\Pi, A, B)$, donde

1. $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$ es el vector de probabilidades iniciales, esto es, la probabilidad de que el estado i sea el primero en la secuencia: $\pi_i = P(q_1 = s_i)$, $i \in [1, ..., N]$, siendo q_1 el estado en el instante 1.

2. $A = [a_{ij}]$ es la matriz de probabilidades de transición. Cada término representa la probabilidad de transición del estado i en el instante t al j en el instante $t+1$: $a_{ij} = P(q_{t+1} = s_j | q_t = s_i)$, $i, j \in [1, ..., N]$.

3. $B = [b_{ik}]$ es la matriz de probabilidades de observación, esto es, la probabilidad de que la observación k se produzca en el estado i en el instante t: $b_{ik} = P(o_t = v_k | q_t = s_i)$, $i \in [1,...,N]$, $k \in [1,...,M]$, siendo o_t la observación en el instante t.

Las siguientes restricciones se aplican para las probabilidades de un HMM:

$$\sum_{i=1}^{N} \pi_i = \sum_{\substack{j=1 \\ \forall i}}^{N} a_{ij} = \sum_{\substack{k=1 \\ \forall i}}^{M} b_{ik} = 1$$

El espacio de observación $V = \{v_1,...,v_M\}$ se refiere a las características que representan cada observación del sistema, *p.ej.*, puertos y direcciones IP de un paquete en el caso de tráfico de red, coeficientes cepstrales de una ventana temporal en reconocimiento del habla, etc.

4.1.2. Clasificación basada en HMM

La clasificación con HMM implica resolver dos cuestiones: la *decodificación* y el *entrenamiento*. Decodificar se refiere a encontrar la secuencia de estados $Q = q_1,...,q_T$ del modelo λ asociada a la secuencia $O = o_1,...,o_T$ observada. Para esto se utiliza el algoritmo de Viterbi:

$$Q = \underset{Q'}{argmax}\ P(Q'|O,\lambda) = \underset{Q'}{argmax} P(O|Q',\lambda)$$

donde

$$P(O|Q',\lambda) = \pi_{q_1} b_{q_1 o_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t o_t} \tag{4.1}$$

Por otro lado, el entrenamiento pretende estimar los parámetros del modelo λ, es decir, Π, A y B. Aunque se pueden encontrar multitud de procedimientos de entrenamiento en la literatura, el algoritmo más utilizado para ello es Baum-Welch.

Para reducir el tamaño del espacio de observación y, por tanto, la complejidad de la parametrización de las observaciones, se utilizan técnicas de cuantización vectorial (VQ). Dos aspectos a tener en cuenta en la cuantización vectorial son:

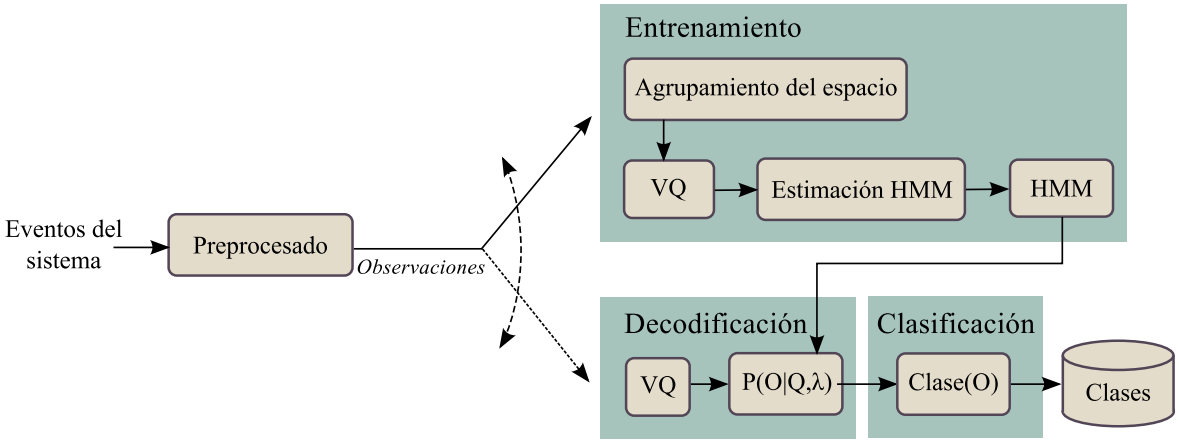


Figura 4.1: Esquema general de un procedimiento de clasificación basado en HMM.

1. En primer lugar se utiliza un algoritmo de *clustering* para cuantizar y, por tanto, reducir el tamaño del espacio de observación. Para esto, el algoritmo K-medias es ampliamente aceptado y utilizado.

2. Una medida de distancia es necesaria para determinar la proximidad de dos vectores de observación dentro del proceso de VQ. Aunque existen algunas, las distancias euclídea y de Mahalanobis son las más extendidas.

Resumiendo, una clasificación basada en HMM sigue el esquema general presentado en la Figura 4.1:

1. Cada evento del sistema se parametriza utilizando un vector multidimensional, obteniéndose así las observaciones, O, de entrada del sistema.

2. Inicialmente se realiza una etapa de entrenamiento a partir de una base de datos de observaciones previamente cuantizadas. El objetivo principal perseguido con el entrenamiento es la estimación de los parámetros que definirán el HMM.

3. Una vez completado el entrenamiento del sistema, la clasificación de cada secuencia de observaciones cuantizadas se realiza como sigue:

   a) La secuencia de observaciones es decodificada de acuerdo al algoritmo de Viterbi (Ecuación (4.1)), obteniendo la probabilidad asociada a dicha secuencia.

   b) Esta probabilidad es comparada con un umbral de detección dado, P_{th}, de modo que la secuencia es considerada como perteneciente al modelo si la probabilidad supera el umbral.

$$Clase(O) = \begin{cases} \lambda, & si\ P(O|Q,\lambda) \geq P_{th} \\ no\ \lambda, & en\ otro\ caso \end{cases} \tag{4.2}$$

4.2. Sistema de clasificación

En lo que sigue se describe la propuesta de clasificación de tráfico P2P basada en el uso de modelos de Markov. Se sugiere cómo construir un sistema de detección capaz de discernir, para una traza de tráfico dada, cuál de los flujos analizados por el sistema corresponde a un servicio o protocolo específico.

Se debe resaltar que para definir completamente un HMM es necesario especificar:

- Las características que serán utilizadas en la etapa de parametrización.
- El proceso VQ, esto es, el algoritmo de cuantización vectorial a utilizar, el número de grupos del espacio de observación y la distancia utilizada como métrica.
- El número de estados y la topología del HMM. Como se indica en [57], existen multitud de aproximaciones para aprender la estructura de los HMMs.

Los detalles de la implementación del sistema de detección propuesto se presentan a continuación.

4.2.1. Observaciones del modelo: preprocesado

La aproximación de clasificación de tráfico presentada se basa en la identificación de flujos. Un flujo se define (como se comentó en el Capítulo 2), de acuerdo con [46], como el tráfico identificado por la tupla `<IP-origen,puerto-origen,IP-destino,puerto-destino,protocolo>`, donde origen y destino son intercambiables para permitir tráfico bidireccional. Debido a que el objetivo de nuestro sistema de detección son los protocolos de aplicación, y más específicamente los protocolos P2P, sólo nos centraremos en paquetes con información de alto nivel.

Cada flujo, y por tanto el modelo HMM asociado, es descrito como una secuencia de paquetes, siendo cada uno de estos paquetes una observación en nuestro sistema. Como se especificó en la Sección 4.1, las observaciones entrantes del sistema de detección pasan por un preprocesado de tráfico. En nuestra propuesta, el preprocesado se divide en tres módulos (ver Figura 4.2): *(i)* parametrización, *(ii)* normalización y *(iii)* cuantización vectorial.

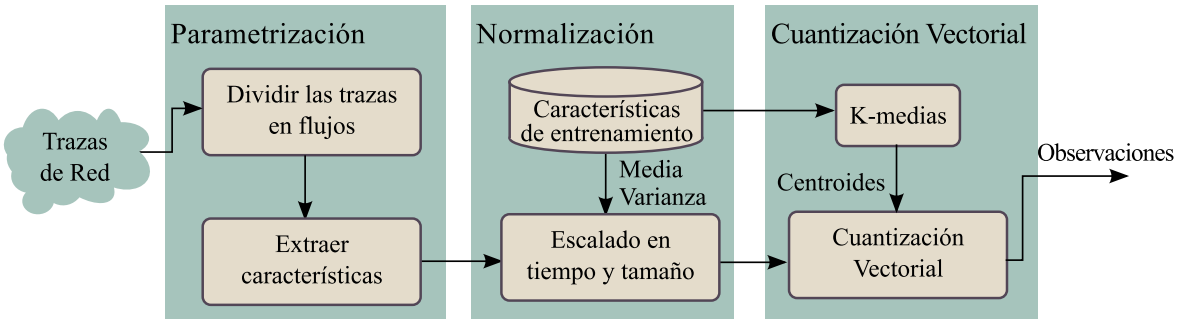


Figura 4.2: Módulos para el preprocesado del tráfico.

Parametrización

La elección del vector de características que describirá el tráfico de red es clave e influye directamente en los resultados de detección del sistema.

En primer lugar, es deseable que las características seleccionadas sean independientes del modo de transmisión de la información. Esto permite la detección de flujos tanto encriptados como no encriptados. En segundo lugar, las características deben ser lo más representativas posible para describir el servicio con mayores garantías de éxito en referencia a la posterior detección. Teniendo esto en consideración, las características seleccionadas son:

1. *Tiempo entre llegadas, (itime):* se define para un paquete como la diferencia, en segundos, entre el tiempo de llegada de éste y la llegada del paquete previo en ese flujo.

2. *Tamaño del contenido, (psize):* es el tamaño, en bytes, de los datos del segmento que viaja en un paquete. Nótese que no se consideran las cabeceras correspondientes a las capas inferiores a la capa de aplicación.

3. *Cambio de dirección, (chdir):* esta característica toma valor '1' para un paquete que viaja en sentido inverso al del paquete previo en el flujo, y '-1' en el caso contrario.

La elección de estas características y no otras ha sido el resultado de un estudio del comportamiento de los flujos de red. En primer lugar, es común que los mensajes que definen un servicio tengan comprendido su tamaño dentro de un rango. Así, se asume que la característica *psize* puede ayudar a caracterizar los mensajes de diferentes protocolos. En cuanto a la característica *itime*, existe también un gran número de protocolos cuyos mensajes requieren una respuesta desde el otro extremo de la comunicación. En estos casos, el tiempo que transcurre hasta que se recibe una respuesta puede ser una característica clave para identificar un protocolo determinado. Nótese, que estas dos características son utilizadas con frecuencia en el campo de clasificación de tráfico [44] [45].

Por lo que sabemos, la tercera característica, *chdir*, no ha sido utilizada hasta la fecha para caracterizar paquetes en el campo de clasificación de tráfico. Algunos autores, como Wright en [44], utilizan la dirección de un paquete en lugar del cambio de dirección. La dirección en este caso podría ser de cliente a servidor o de servidor a cliente. Sin embargo, un problema del uso de esta característica es que no puede describir convenientemente el comportamiento de protocolos P2P en los que no existen los roles de cliente o servidor. Para este propósito es más adecuado utilizar el cambio de dirección, que no precisa el conocimiento de qué extremo de la comunicación actúa como servidor o cliente.

Normalización

Este es el segundo módulo del preprocesado. En nuestro sistema, la variación de los valores de las características del modelo de Markov describe, entre otros aspectos, el protocolo a modelar. Si el rango dinámico es muy diferente entre las características, una variación en el valor de una característica con mayor rango dinámico influirá más en el modelo que la misma variación proporcional en una característica de menor rango dinámico.

La característica *psize* puede tomar valores mayores que 0 y menores que el tamaño de la MTU (*Maximum Transfer Unit*) sin considerar las cabeceras de capas inferiores a la de aplicación. La variable *itime* ha de ser mayor que 0 y *chdir* sólo puede tomar los valores '1', si existe cambio de dirección, o '-1', en otro caso.

El objetivo del módulo de normalización es unificar el rango dinámico de las características que representan un paquete. El rango seleccionado es [-1,1], con lo que *chdir* no debe ser modificado. Con respecto a *itime*, primero realizamos una transformación logarítmica para reducir el rango dinámico [59]. Posteriormente, se normaliza el valor resultante realizando un desplazamiento en media y un autoescalado, $v_s = (v_o - \mu)/\sigma$, donde v_s es el valor escalado, v_o el original y μ y σ la media y desviación estándar de los valores a escalar respectivamente. Para el caso de la característica *psize* sólo se aplica un desplazamiento en media y un autoescalado.

Para aplicar la normalización descrita es necesario extraer previamente la media y desviación estándar de los datos. Estos valores se calculan en la fase de entrenamiento sobre los datos dispuestos al efecto.

Cuantización vectorial

Tras la parametrización y normalización, se lleva a cabo un proceso de cuantización sobre los vectores de características obtenidos para cada observación (paquete) de un flujo dado. La cuantización vectorial utilizada en el sistema propuesto se basa

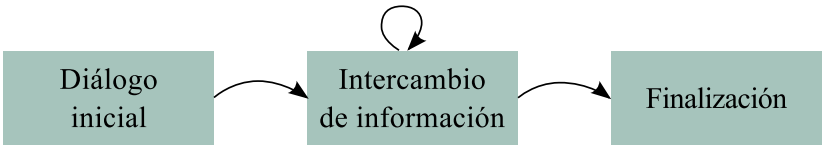


Figura 4.3: Etapas del modelo genérico de Markov para describir un protocolo/servicio.

en el algoritmo K-medias [60]. De este modo, todas las posibles combinaciones de los valores de las tres características se representan por K vectores. Estos vectores son los centroides correspondientes a los K *clusters* derivados del agrupamiento de los datos contenidos en la base de datos de entrenamiento. La métrica utilizada en dicho agrupamiento es la distancia euclídea.

En consecuencia, cada vector de características (cada paquete) quedará representado por el índice del centroide más cercano y las observaciones consideradas en el HMM serán, por tanto, una secuencia de índices de centroides derivada de la cuantización vectorial de cada paquete entrante.

4.2.2. Estructura del HMM

La parte fundamental de la metodología de detección utilizada es el uso de un modelo de Markov para describir el tráfico correspondiente al protocolo/servicio objetivo. La definición de los parámetros concretos del modelo de Markov, es decir, las posibles observaciones del modelo, los estados y las transiciones entre estados, dependerá del protocolo a detectar y, por tanto, será especificado para cada protocolo/servicio en particular.

Pese a lo anterior, a continuación se propone un modelo genérico para representar las fases de la mayoría de los servicios de comunicación. Éste consta de tres etapas (Figura 4.3):

1. *Diálogo inicial*

   Representa el inicio de la comunicación, en el que los miembros de la misma intercambian cierta información que será utilizada con posterioridad. Por ejemplo, en el caso de SSH, para iniciar la comunicación tras establecer la conexión TCP, es necesario llevar a cabo una solicitud de autenticación de usuario y su correspondiente respuesta.

2. *Intercambio de información*

   En esta fase los participantes transmiten la información objetivo de la comunicación: transferencia de archivos, descarga de una página web, etc. Es esperable

que esta fase contenga la mayor parte de la información intercambiada entre los extremos.

3. *Finalización*

   Después de la fase de intercambio de información es usual que algunos protocolos envíen paquetes indicando la finalización de la comunicación.

Un aspecto importante a tener en cuenta en el diseño de la estructura del modelo de Markov es la variabilidad en el número de observaciones (paquetes) de los flujos de un servicio/protocolo. Este hecho, como puede verse en la Figura 4.3, es abordado definiendo la etapa de *intercambio de información* como "reejecutable". Por tanto, esta etapa es capaz de describir transferencias de datos y señalización de cualquier número de paquetes. Por ejemplo, en el caso del protocolo SSH todos los paquetes intercambiados por la ejecución de un comando remoto podrían ser representados por una única etapa de intercambio de información. La ejecución de otro comando remoto implicaría la reejecución de esta etapa.

Finalmente, las etapas de *diálogo inicial* y *finalización* se ejecutan una sola vez, dado que se espera que su ejecución sea única en una comunicación.

4.2.3. HMM para la detección de los flujos eDonkey

Con el objetivo de explorar el funcionamiento del sistema de detección propuesto se elige el protocolo eDonkey como un caso de estudio concreto. Para esta tarea se parte del modelo genérico presentado con anterioridad y se especifican los estados y transiciones entre ellos de las diferentes etapas del modelo genérico que permitan representar de la mejor forma posible el comportamiento del protocolo eDonkey.

El protocolo eDonkey presenta algunas características que lo hacen especialmente adecuado para resaltar la potencialidad de nuestra propuesta. En primer lugar, al ser un protocolo P2P, no existen clientes ni servidores. Además, este protocolo se utiliza para compartir archivos, lo que implica que existen flujos de datos y también de señalización. Y finalmente permite dos tipos de comunicación: ofuscada y sin ofuscar. Estas características conllevan una elevada variabilidad, lo que en suma dificulta su modelado.

El protocolo eDonkey utiliza tanto UDP como TCP, pero los flujos TCP representan más de un 95 % de los flujos eDonkey y más de un 99 % de los bytes transmitidos[1]. Por tanto, centramos la detección en los flujos TCP de eDonkey.

[1] Estos porcentajes se extrajeron de un estudio de las trazas descritas en la Sección 4.3.

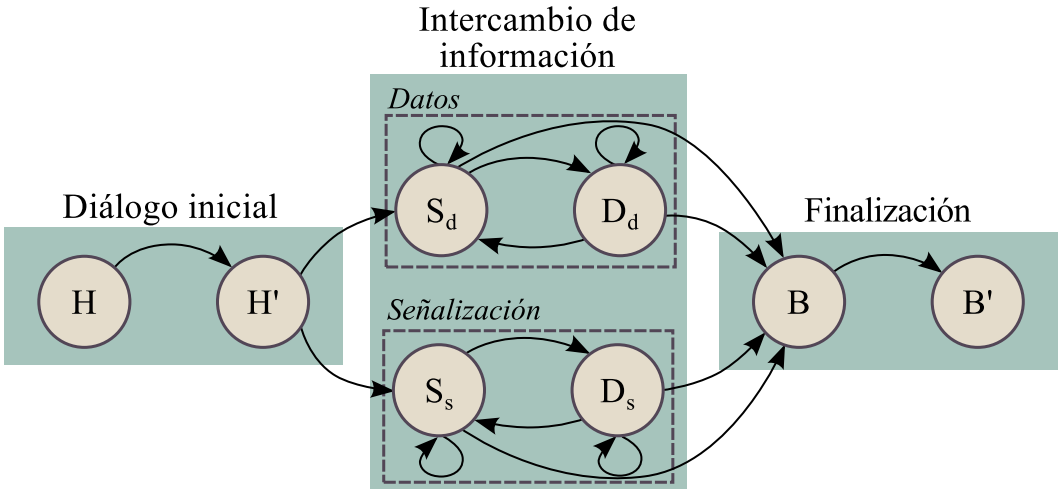


Figura 4.4: HMM para modelar el protocolo eDonkey.

La especificación de los estados del modelo de Markov para la detección de flujos TCP del protocolo eDonkey puede verse en la Figura 4.4. Las etapas primera y última del modelo genérico, es decir, las etapas de diálogo inicial y finalización, están compuestas de dos estados con una única transición entre ellos. Esto se debe al propio funcionamiento del protocolo eDonkey. Los mensajes intercambiados en el diálogo inicial son `Hello` (representado como estado H) y `Hello_answer` (estado H'). Por medio de estos dos mensajes los nodos eDonkey intercambian sus identificadores únicos. La finalización de la comunicación se suele componer de un mensaje de solicitud de entrada en la cola de servicio (estado B, mensaje `Start_upload_request`) y la respuesta del extremo opuesto indicando la posición que ocupa la petición en dicha cola (estado B', mensaje `Queue_rank`). La razón de esta transición es que el protocolo eDonkey pretende que el mayor número de usuarios puedan acceder al mayor número de recursos. De modo, aunque un *peer* A comience a descargar de otro B un recurso, B parará de compartir con él tras enviarle varias partes del recurso solicitado para que así otros *peers* puedan beneficiarse de esta compartición. Por esto mismo, el *peer* A terminará la comunicación solicitando de nuevo la entrada en la cola de conexiones del *peer* B. Para más detalles acerca de estos mensajes y el resto de los mensajes del protocolo eDonkey, acuda a [61].

En cuanto a la etapa de *intercambio de información*, no se puede identificar un solo estado para cada paquete en el flujo ya que el número de paquetes por flujo es variable. Así, se definen dos caminos posibles: uno para representar los flujos que denominamos de *datos* (número elevado de paquetes, con un contenido elevado), y otro para flujos de *señalización* (número reducido de paquetes en el flujo, con contenido reducido). Cada uno de estos dos caminos está compuesto de dos estados, uno para representar los paquetes de datos (estado D) y otro para los de señalización (estado S). Esto se debe a que tanto los flujos de señalización como los de datos se

componen de paquetes de ambos tipos, esto es, de señalización y de datos. Como se mencionó con anterioridad, las transiciones definidas en esta etapa dan la posibilidad de ser ejecutadas tantas veces como sea necesario para recoger todos los paquetes transmitidos en la etapa de intercambio, pudiendo pasar a la etapa final tanto desde los estados de señalización como desde los de datos.

4.3. Evaluación experimental

En esta sección se describe la evaluación experimental realizada para validar el sistema de clasificación de tráfico basado en HMM y adaptado para la clasificación de flujos del protocolo eDonkey. La estrategia de evaluación se compone de dos partes: (*i*) análisis de comunicaciones reales del protocolo eDonkey para comprobar si el modelado se ajusta al comportamiento del protocolo, y (*ii*) evaluación de los resultados obtenidos por el sistema de detección.

4.3.1. Descripción del entorno experimental

Se han utilizado dos grupos de trazas de tráfico real de red para llevar a cabo la evaluación del sistema propuesto. Las características principales de estas trazas se describen a continuación.

- *Trazas de tráfico eDonkey, eD*

  Este conjunto está compuesto por el tráfico eDonkey generado por un nodo durante 45 días, 5 horas cada día (desde el 7 de julio al 9 de septiembre de 2011). La aplicación utilizada para compartir archivos mediante eDonkey fue aMule 2.2.6 [62]. El cliente se conectó al servidor eDonkey denominado *se-Master Server 1*. Para cubrir un amplio rango de *peers* diferentes de la red eDonkey se seleccionaron archivos para descargar y compartir con alta, media y baja popularidad.

  Las trazas se componen de 240.851 flujos TCP y 7.003 UDP, todos ellos pertenecientes al protocolo eDonkey. Entre los flujos TCP, 22.409 estaban ofuscados. Todas estas comunicaciones se llevaron a cabo entre más de 12.000 direcciones IP diferentes, con una transferencia de más de 20 GB de descarga y 25 GB de subida. A diferencia de la traza EC del capítulo anterior esta traza es mucho más extensa y contiene solamente tráfico eDonkey.

- *Trazas de un troncal de una universidad de Oriente Medio, TU*

  Este conjunto de trazas de tráfico fue utilizado en los experimentos descritos en el capítulo anterior. Contiene todo el tráfico generado durante 48 horas

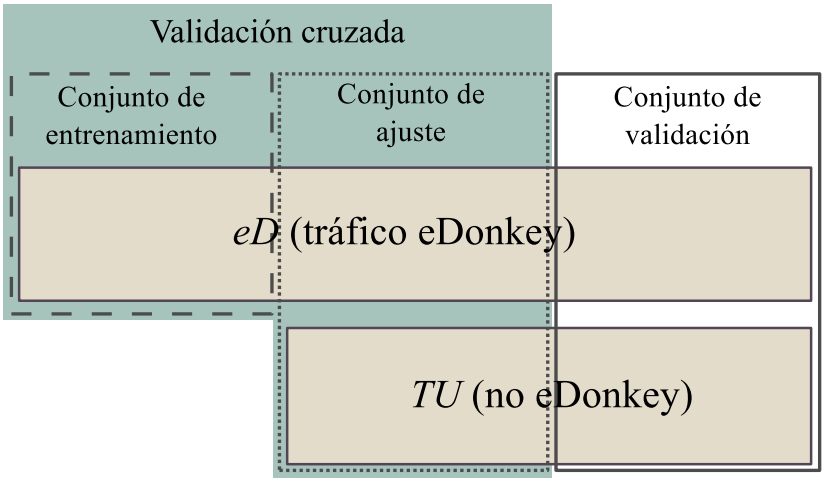


Figura 4.5: Partición de los conjuntos de datos en: entrenamiento, ajuste y validación.

(noviembre de 2010) en una universidad de Oriente Medio. En resumen, hay alrededor de 73.000 direcciones IP y 300 millones de paquetes transmitidos. Los protocolos más comunes en ella son: BitTorrent, HTTP, DNS, SSL y RTP (*Real-time Transport Protocol*). No se detectó ningún paquete del protocolo eDonkey.

Como puede verse en la Figura 4.5, ambos conjuntos de datos se unifican en uno solo, que por un lado contiene tráfico eDonkey etiquetado (*eD*) y por otro lado tráfico que no contiene flujos eDonkey (*TU*).

Para entrenar el modelo se necesitan flujos del protocolo eDonkey, así que se utiliza un subconjunto específico de la traza *eD*. Para el ajuste del sistema se usa un conjunto de datos que contiene un subconjunto diferente de *eD* y un subconjunto de *TU*. Como el conjunto de ajuste está completamente etiquetado, se puede extraer la tasa de aciertos del sistema en función del porcentaje de flujos detectados del subconjunto *eD*. Por otro lado, todos los flujos detectados como eDonkey dentro del subconjunto *TU* representan la tasa de falsos positivos del sistema.

Se lleva a cabo también un proceso de validación cruzada de la siguiente forma:

1. El conjunto de datos de ajuste más el conjunto de entrenamiento es dividido en 10 partes de igual tamaño.
2. Uno de estos grupos se reserva para ajuste (realizando un proceso de detección) y el resto se utiliza para entrenar el sistema.
3. Este proceso se realiza 10 veces y, para cada una de ellas, un grupo diferente es utilizado para ajustar el sistema y el resto para entrenarlo.

Tabla 4.1: Mensajes eDonkey más frecuentes en cada etapa del modelo de Markov.

Etapa	Mensaje más frecuente	Porcentaje
Diálogo Inicial	`Hello` y `Hello_answer`	99,89 %
Terminación	`Start_upload_request` y `Queuerank`	96,24 %

4. De esta forma, se obtienen 10 valores diferentes de tasa de aciertos y de falsos positivos considerándose la media de éstos como el valor real del sistema.

Por último, una parte de los conjuntos de datos *TU* y *eD* que no ha intervenido ni en el entrenamiento ni en el ajuste se reserva para realizar el proceso de validación del que se puede extraer la curva ROC del sistema de detección.

4.3.2. Análisis de flujos reales de eDonkey

En primer lugar pretendemos inspeccionar si las trazas reales de eDonkey se comportan tal y como esperábamos. Para esto, se ha extraído el tipo de mensaje de los dos primeros y dos últimos paquetes de 240.851 flujos TCP de eDonkey. Se utilizó una modificación del cliente aMule para ser capaces de monitorizar el contenido de todos los paquetes, incluyendo los de flujos ofuscados.

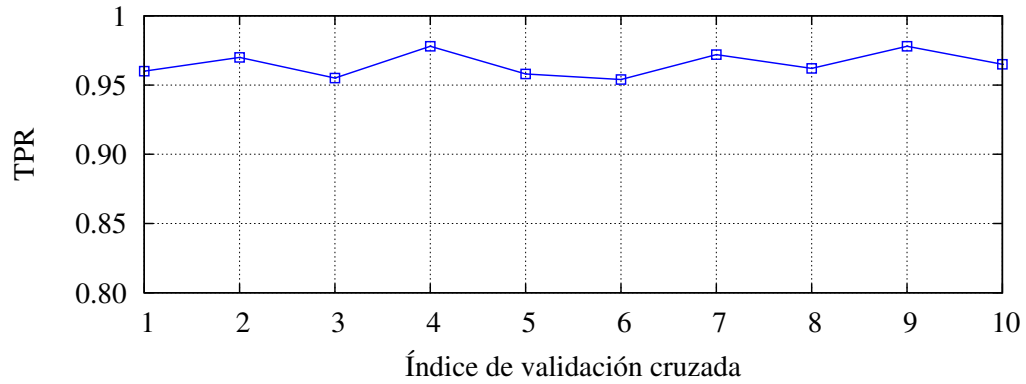
Como resultado de esta inspección (ver Tabla 4.1) podemos afirmar que un 99,89 % de los flujos comienzan con mensajes `Hello` y `Hello_answer`. Adicionalmente, el 96,24 % de los flujos finalizan con los mensajes `Start_upload_request` y `Queue_rank`. Esto se debe a que un *peer*, al recibir varias partes de un recurso de otro, vuelve a pasar a la cola de espera para posibilitar que otros *peers* descarguen el recurso solicitado. Se ha comprobado también que este porcentaje no es del 100 % debido a la existencia de flujos que acaban de forma abrupta.

A la luz de estos resultados se confirma que las hipótesis asumidas para definir la estructura del HMM para modelar eDonkey son razonables.

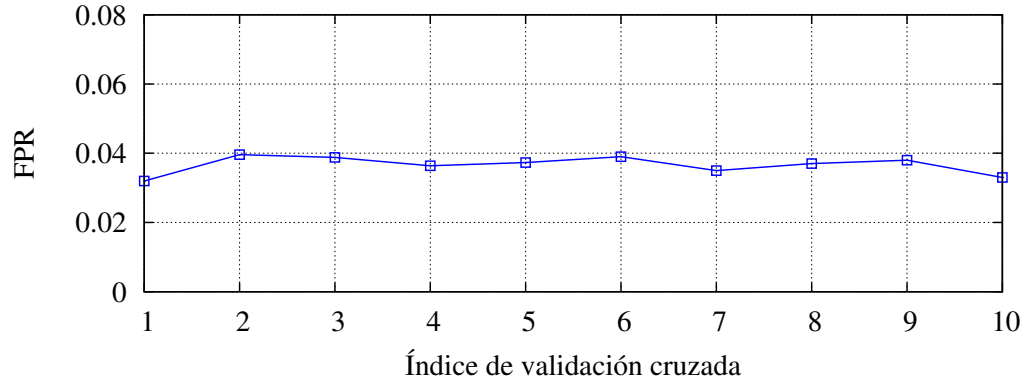
4.3.3. Resultados de identificación

A continuación estamos interesados en comprobar las tasas de detección alcanzadas por el sistema propuesto. Para esto, se entrena el modelo como se explicó anteriormente y se obtienen los resultados de detección.

Para preprocesar las observaciones se ha utilizado K-medias con un valor $K = 32$. Así, cada paquete pasa a ser representado por el índice del centroide (de 1 a 32) más



(a)



(b)

Figura 4.6: Tasa de (a) verdaderos positivos y (b) falsos positivos, obtenida para cada evaluación en el proceso de validación cruzada.

cercano. La métrica de distancia utilizada en el proceso de cuantización vectorial es la distancia euclídea.

En la Figura 4.6 se muestran los resultados del proceso de validación cruzada. En la subfigura (a) se puede ver la tasa de verdaderos positivos, TPR, y en la (b) la tasa de falsos positivos, FPR, para cada una de las 10 evaluaciones implicada en la validación cruzada. En todos los casos se obtiene más de un 95% de aciertos y alrededor de un 4% de falsos positivos.

Se ha realizado un estudio de la dependencia de la calidad de la detección en función de la elección del umbral de detección, P_{th} (ver Ecuación (4.2)). Para ello se ha evaluado la base de datos de validación barriendo valores de probabilidad en el intervalo [0,01-0,4] para el umbral P_{th}. Los resultados de este estudio permiten representar la curva ROC del sistema, la cual se observa en la Figura 4.7. Como puede verse, existe un amplio rango de valores de la TPR (0,9 a 0,95) en los que la FPR es

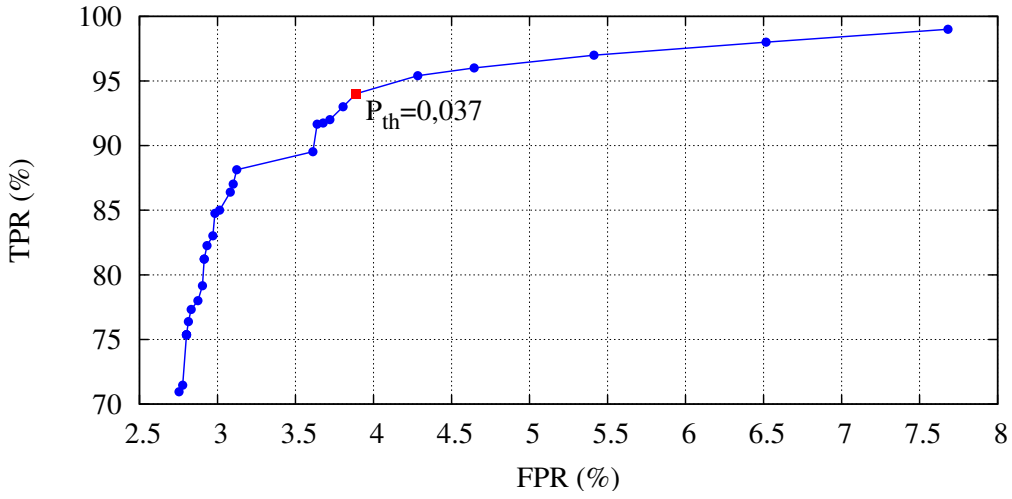


Figura 4.7: Curva ROC del sistema de detección para el protocolo eDonkey.

Tabla 4.2: Tasa de aciertos del sistema, TPR.

	Tasa de verdaderos positivos
Flujos eDonkey sin ofuscar	96,6 %
Flujos eDonkey ofuscados	94,2 %

reducida (de 0,036 a 0,038). Nótese que nuestro sistema generaliza bien, ya que los resultados obtenidos en la validación son muy cercanos a los que se obtuvieron en el ajuste del sistema.

Como se mencionó, *eD* contiene tanto flujos ofuscados como no ofuscados. Particularizando en la tasa de aciertos obtenidos en estos casos se obtuvo un 96,6 % para el caso de flujos ofuscados, frente a un 94,2 % para los no ofuscados (ver Tabla 4.2). Estos resultados demuestran que el sistema presentado es independiente de la existencia o no de encriptación en el protocolo objetivo. Esto es lógico ya que el sistema de clasificación propuesto utiliza una parametrización que no depende de la encriptación.

Finalmente, se lleva a cabo una evaluación adicional de nuestro sistema, que consiste en el estudio de los resultados de detección obtenidos cuando no existen flujos del protocolo eDonkey en las trazas analizadas, lo que implica que todo flujo detectado representará un falso positivo.

Se escogen tres protocolos presentes en la traza *TU*: *(i)* HTTP, *(ii)* protocolos para *streaming* multimedia (RTP) y *(iii)* protocolos para la compartición de archivos mediante P2P. El protocolo HTTP se ha seleccionado por su amplio uso en Internet. El protocolo RTP se utiliza típicamente para transmitir *streaming* multimedia de

Tabla 4.3: Tasa de falsos positivos para la detección de eDonkey cuando se analizan otros protocolos.

	Flujos totales	**Tasa falsos positivos**
HTTP	710.037	4,0 %
Streaming Multimedia (RTP)	58.509	0,1 %
Compartición de archivos P2P	215.203	1,7 %

modo que se envían ráfagas de gran tamaño que viajan en una misma dirección. Estamos especialmente interesados en estos protocolos porque su comportamiento puede asemejarse al de eDonkey, en el que se envían ráfagas de paquetes cargados de contenido. Por la misma razón, se considera el tercer grupo, protocolos de compartición de archivos mediante P2P distintos de eDonkey. Los protocolos presentes en la traza *TU* de este tipo son Gnutella y BitTorrent.

Los resultados obtenidos en esta evaluación se muestran en la Tabla 4.3. En ésta se pueden ver las tasas de falsos positivos en función del protocolo original de los flujos evaluados. Nótese que nuestro sistema de detección es capaz de diferenciar tanto flujos de aplicaciones de compartición de archivos mediante P2P como de *streaming* multimedia con una tasa de falsos positivos muy reducida. Finalmente, HTTP es también diferenciado de eDonkey con una tasa de falsos positivos bastante aceptable.

4.3.4. Comparación de los resultados de clasificación obtenidos

A continuación se comparan los resultados de clasificación obtenidos con el presente sistema con respecto a otros sistemas de la literatura.

Los autores en [43] utilizan la información de los paquetes de control como características para representar las observaciones de su modelado y el sistema presentado depende del número de paquetes de control en el flujo. Los resultados obtenidos presentan una elevada tasa de falsos positivos, alrededor de un 10 %. Nuestra propuesta es independiente del número de paquetes de un flujo y alcanza mejores resultados de detección.

Los trabajos [44] y [37] alcanzan tasas de detección inferiores a nuestro sistema.

El modelo propuesto en [45] es entrenado con pocos flujos, por ejemplo, eDonkey es entrenado con 109 flujos y evaluado con 82. En nuestro caso, el entrenamiento y evaluación utilizan más de 240.000 flujos y el sistema de detección es capaz de alcanzar una mejor tasa de verdaderos positivos con una tasa similar de falsos positivos.

Este buen comportamiento de detección es destacado con respecto a otras aproximaciones existentes [15] [29] [38] e incluso con respecto a la aproximación propuesta en el Capítulo 3 que alcanzaba tasas de detección TPR y FPR de 77,53 % y 1,139 % respectivamente para el caso de WCFD y de 86,10 % y 1,678 % para el caso de URND.

En resumen, los resultados de detección obtenidos demuestran la bondad de nuestra aproximación en comparación con las presentes en la literatura.

4.4. Conclusiones

En este trabajo se introduce una metodología basada en el uso de modelos de Markov para determinar la provisión de ciertos servicios/protocolos fundamentado en la clasificación de tráfico. En primer lugar, se contribuye con un HMM genérico a nivel de flujo y cuyas observaciones son los paquetes del flujo. Tras esto, se particulariza el modelo para representar las comunicaciones del protocolo eDonkey como caso de estudio. Para ello se discuten los aspectos clave involucrados: cuantización vectorial, normalización, estructura del sistema, etc.

Para validar la propuesta presentada se realiza una experimentación con tráfico eDonkey y no-eDonkey. Los resultados obtenidos muestran elevadas tasas de detección junto a reducidas tasas de falsos positivos. El sistema mantiene esta buena actuación incluso cuando los flujos de eDonkey están ofuscados o los protocolos evaluados son también de tipo P2P de compartición de archivos como BitTorrent o Gnutella.

Además, se ha comprobado que el sistema propuesto es robusto a las comunicaciones ofuscadas ya sea por un cambio en el puerto de origen/destino como por el uso de cifrado. Este sencillo hecho constituye una aportación significativa de nuestra propuesta. Por último, cabe resaltar la bondad de los resultados obtenidos con respecto a los reportados en la literatura para otros métodos de clasificación de tráfico.

Parte II

Estudio y detección de *botnets*

Capítulo 5

Revisión y taxonomía de la investigación en *botnets* a través de su ciclo de vida

Como consecuencia del estudio de los métodos de clasificación de tráfico P2P, en este trabajo se ha identificado un tema de especial relevancia en relación a la seguridad de estas redes, y que ha constituido el objeto de esta segunda parte del presente documento. En esta parte se estudiarán las *botnets*, una amenaza que ha centrado recientemente el interés de la comunidad investigadora provocando un aumento exponencial del número de publicaciones en estos años.

En respuesta a este elevado número de contribuciones, en el capítulo presente se propone una taxonomía de la investigación en *botnets* y una revisión de las mismas, con el objetivo de dar una visión general de este campo. La taxonomía presentada se basa en una nueva propuesta de ciclo de vida de las *botnets*, que se construye como una secuencia de etapas que toda *botnet* necesita superar para alcanzar su objetivo final, un ataque satisfactorio. Una de las conclusiones extraídas de la creación de esta taxonomía es que para neutralizar una *botnet* se debe interrumpir al menos una de las etapas de este ciclo de vida, lo que impedirá que la *botnet* alcance su siguiente etapa y, por ende, su objetivo final.

Las capacidades de la taxonomía propuesta son comprobadas por medio de una revisión de los trabajos de investigación más relevantes sobre *botnets*, resultando de elevada utilidad en la comprensión y organización de las diferentes contribuciones de esta área. Adicionalmente, se pretende aportar una perspectiva más clara sobre las cuestiones aún sin resolver en la línea de investigación sobre defensas frente a *botnets*.

Tabla 5.1: Pérdidas anuales provocadas por crimen cibernético [63]

Concepto	**Año**	**Pérdidas anuales**
Fraude bancario en U.K.	2008	36,5 millones de libras
Robos de identidad en E.E.U.U.	2006	2.800 millones de dólares
Daños en Europa provocados por *malware*	2006	9.300 millones de euros

El resto del capítulo se estructura de la siguiente forma. En primer lugar, en la Sección 5.1 se indica la problemática actual que suponen las *botnets* y se motiva la necesidad de una revisión organizada de la investigación hasta la fecha. En la Sección 5.2 se presentan los conceptos fundamentales en el campo de las *botnets*. Posteriormente, en la Sección 5.3 se introduce y describe la taxonomía propuesta, mientras que en la Sección 5.4 se presenta una selección de los artículos de investigación más relevantes en el campo de las *botnets*, organizados de acuerdo a la taxonomía presentada. Algunos de los retos aún por abordar en la investigación en el campo se indican en la Sección 5.5 y finalmente las conclusiones principales de este capítulo se resumen en la Sección 5.6.

5.1. Motivación

Las *botnets* representan en la actualidad una de las amenazas más importantes a la ciberseguridad. El término *botnet* se utiliza para definir una red de máquinas infectadas, llamadas *bots*, que son controladas por un operador humano, comúnmente conocido como *botmaster*. Los *bots* se utilizan para una gran variedad de actividades maliciosas y dañinas contra sistemas y servicios, incluyendo ataques DoS, distribución de *spam*, distribución de *malware*, *phishing* y fraude del *click* [6] [7]. Como ejemplo del impacto de los ataques a la ciberseguridad, en la Tabla 5.1 se indican las pérdidas anuales debidas a este tipo de ataques en distintos lugares del mundo.

El beneficio económico es el motivo principal que impulsa a los *botmasters* a diseñar e implementar nuevas *botnets*. Anunciando una *botnet* en la red es posible conseguir grandes sumas de dinero. Un ejemplo del beneficio de alquilar los servicios ilegítimos de una *botnet* puede ser el caso de Jeanson Ancheta, un joven de 21 años miembro de un grupo de *hackers* denominado "Botmaster Underground". Jeanson consiguió más de 100.000 dólares de diferentes compañías de publicidad que le pagaron por instalar un *adware* (*malware* de publicidad), específicamente diseñado para este caso, en más de 400.000 PCs vulnerables previamente infectados por la *botnet* de Ancheta [64].

El peligro creciente que suponen las *botnets* fue también apuntado por Vinton Cerf, uno de los "padres de Internet", que estimó que entre 100 y 150 de los 600

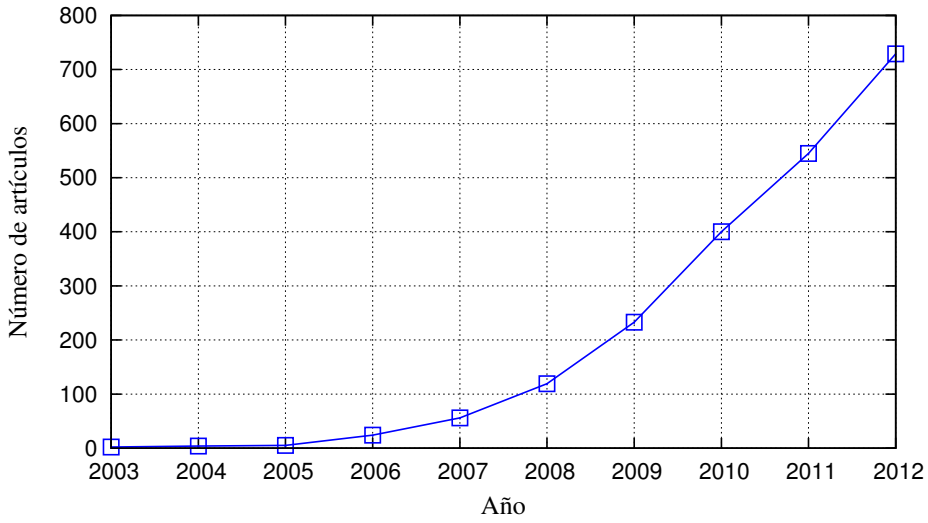


Figura 5.1: Suma acumulada del número de publicaciones en el campo de las *botnets* en los últimos años (datos tomados de la plataforma Scopus [11]).

millones de equipos conectados a Internet formaban parte de una *botnet* [8]. En esta misma línea, entre el 1 de julio y el 31 de diciembre de 2007, Symantec detectó una media de 61.940 *bots* activos por día [9]. Esta cifra supuso un incremento del 17 % con respecto a los primeros seis meses de 2007. Symantec también detectó alrededor de 5 millones de *bots* diferentes durante este periodo de tiempo y 4.091 servidores C&C (*Command and Control*). En un estudio adicional realizado por McAfee [10] entre el 1 de enero y el 31 de marzo de 2009 se descubrieron cerca de 12 millones de nuevas direcciones IP actuando como *bots*, lo cual supone un incremento notable con respecto a las cifras aportadas por Symantec dos años antes.

Como consecuencia del impacto de las *botnets*, el interés en este campo ha crecido enormemente entre la comunidad investigadora. Como se muestra en la Figura 5.1, el número de publicaciones que abordan el problema de las *botnets* ha aumentado exponencialmente en la última década, desde sólo unos pocos en los primeros años hasta casi 200 artículos nuevos en 2012.

En los últimos años se han propuesto algunas taxonomías en el campo de las *botnets* [65] [66] junto a algunas revisiones de la investigación en este campo [6] [67] [68] [69]. Sin embargo, estas taxonomías se centran fundamentalmente en los diferentes aspectos técnicos de las *botnets*, como pueden ser su arquitectura, protocolos de comunicación o técnicas de detección, presentando una taxonomía diferente para cada uno de estos aspectos. De esta forma, aunque estas taxonomías permiten la comprensión de ciertos aspectos particulares de las *botnets*, es complicado alcanzar una visión global del problema a partir de las mismas.

Por esta razón, es preciso definir una taxonomía que aporte una perspectiva global del problema de las *botnets*. En este capítulo se pretende ordenar la gran cantidad

de trabajos de investigación existentes en este campo, resaltando las principales preocupaciones y retos presentados en éstos.

En este contexto, se propone una descripción de las *botnets* desde la perspectiva de su ciclo de vida. Como se detallará extensamente en lo que sigue, el ciclo de vida de una *botnet* comienza con la concepción de la misma, y continúa hasta la consecución de su objetivo final, que consiste de un modo u otro en la ejecución satisfactoria de un ataque concreto. Este ciclo de vida es lineal y está compuesto de un conjunto de etapas que deben completarse durante la evolución de la *botnet* hasta la consecución final del ataque.

El análisis de las *botnets* desde la perspectiva de su ciclo de vida permite entender el proceso de creación, desarrollo, integración y uso de una *botnet*, pero también facilita la organización de la gran cantidad de trabajos que la comunidad investigadora ha realizado con el objetivo de frenar el avance de esta gran amenaza. Así, mediante la taxonomía propuesta se realiza una revisión de los trabajos más relevantes del campo en relación a las distintas etapas que componen su evolución.

Aún más importante es que, como se mencionó con anterioridad, el presente trabajo permite destacar que cualquier técnica de defensa debe ser diseñada con la siguiente idea en mente: "Impedir la ejecución exitosa de cualquiera de las etapas del ciclo de vida de una *botnet* permitirá neutralizar la *botnet* y sus efectos".

5.2. Conceptos fundamentales sobre *botnets*

Como se comentó con anterioridad, una *botnet* es una red de máquinas infectadas bajo el control de un operador humano. Existen dos componentes fundamentales dentro de una *botnet*: los *bots* y el *botmaster*. Las máquinas infectadas son denominadas *bots*, un término derivado de la palabra "robot" y que apunta al hecho de que sigue las instrucciones dadas por el operador humano, el *botmaster*. Éste es el que controla la *botnet* y, por tanto, la utiliza para conseguir su objetivo último, generalmente llevar a cabo un ataque de seguridad contra una víctima.

Una *botnet* es controlada por medio de la transmisión de mensajes denominados mensajes de órdenes y control (*Command and Control*, C&C) entre sus miembros. Para que esto sea posible, el *botmaster* debe establecer previamente ciertos canales C&C para comunicarse con los *bots*. En algunos casos, los *bots* se conectan a un servidor C&C para recibir los mensajes enviados por el *botmaster*. La existencia de este servidor y la arquitectura de las comunicaciones C&C dependen del propio diseño de la *botnet*. Las diferentes alternativas utilizadas hasta la fecha se describen en la Sección 5.3.1. Aparte de los actores fundamentales de las *botnets*, es decir, los *bots* y el *botmaster*, aparecen otros roles importantes:

- *Desarrollador*

  Es la persona o grupo de personas que se encargan de diseñar e implementar el código de la *botnet*. El desarrollador no es necesariamente la misma persona que el *botmaster*, ya que el trabajo de diseño e implementación puede ser subcontratado. En esta línea, existe multitud de *malware* que facilita todas las herramientas necesarias para construir y administrar una *botnet*, comúnmente denominado DIY (*Do-it-Yourself*). Se pueden encontrar multitud de ejemplos: Zeus [70], Twitter [71], Aldi [72], etc.

- *Cliente*

  Existen dos tipos principales de clientes en el campo de las *botnets*. Unos clientes alquilan los servicios de una determinada *botnet* a un *botmaster*. Ejemplos de servicios a alquilar pueden ser: ataques DDoS o distribución de *spam*. Por otro lado, existen clientes que compran la *botnet* en su totalidad, convirtiéndose de esta forma, en *botmasters*. Acto seguido, pueden utilizar la *botnet* para sus propios propósitos.

- *Víctima*

  Este es el sistema, persona o red que constituye el objeto de ataque de la *botnet*. Hay muchos tipos diferentes de víctimas, dependiendo del propósito principal de la *botnet*. Por ejemplo, un usuario que recibe *spam*, alguien a quien se le roba información confidencial, una compañía que pierde millones de dólares como consecuencia de un ataque DDoS, etc.

- *Participante pasivo*

  Es el propietario de un equipo que ha sido infectado por una *botnet* y que, por tanto, se ha convertido en un *bot* de ésta. Este nuevo *bot* llega a ser parte de la *botnet* sin el consentimiento del dueño del equipo infectado. Sin embargo, su participación incluso sin este consentimiento podría incurrir en consecuencias legales. Matthew Bandy, un joven de 16 años, estuvo cerca de ser sentenciado a 90 años de prisión por ser un participante pasivo. Su ordenador como parte de una *botnet* distribuyó sin su consentimiento pornografía infantil [73]. Afortunadamente, se pudo demostrar que Matthew no se encontraba siquiera en su hogar cuando comenzó el tráfico de descarga y compartición de estos recursos ilegales y pudiendo así ser exculpado tras un largo proceso judicial.

Las principales particularidades de las *botnets* que las diferencian de otros tipos de *malware* pueden ser resumidas en dos: (*i*) el *botmaster* puede enviar órdenes a un equipo infectado sin controlar directamente esta máquina y (*ii*) los *bots* actúan de forma coordinada, siguiendo las instrucciones del *botmaster*. La capacidad de controlar un enorme número de *bots* de forma coordinada permite al *botmaster* ejecutar ataques masivos, como DDoS, fraude del *click*, distribución de *spam*, etc.

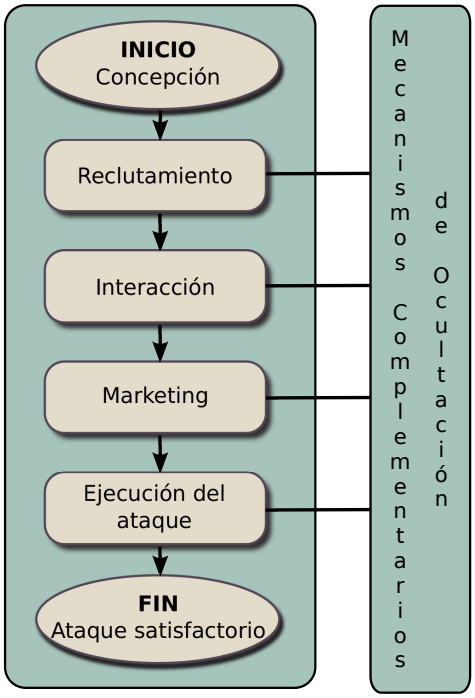


Figura 5.2: Etapas y mecanismos complementarios del ciclo de vida de las *botnets*.

Adicionalmente, las *botnets* se diseñan para intentar ocultar las interacciones mencionadas con anterioridad y así dificultar la detección por parte de los agentes de seguridad. Esta ocultación se lleva a cabo por medio de técnicas como el multi-salto, el cifrado y la ofuscación de los binarios que serán descritas en la Sección 5.3.6.

5.3. Ciclo de vida de las *botnets*

El concepto de ciclo de vida de las *botnets* ha sido propuesto con anterioridad en la literatura especializada, y existen algunos artículos que trabajan en esta línea [6] [74] [7] [75]. Sin embargo, estos estudios ilustran solamente algunos de los procesos involucrados en la operación normal de una *botnet*, no existiendo una aproximación que enfoque este problema desde un punto de vista global, y que a su vez describa las etapas que componen el ciclo de vida y las interacciones entre ellas.

En respuesta a la ausencia de un ciclo de vida completo, se propone a continuación un ciclo de vida para las *botnets*, que va desde su concepción hasta la consecución de su objetivo final. Este ciclo de vida es una secuencia lineal de etapas, es decir, el fin del mismo (el ataque exitoso), se alcanza únicamente tras la ejecución exitosa de todas y cada una de las etapas previas. Por tanto, un fallo en cualquiera de las etapas intermedias impediría que la *botnet* alcanzara su objetivo.

A fin de justificar la hipótesis anterior, a continuación se describen las etapas que componen el ciclo de vida de una *botnet* y los procesos y roles que intervienen en ellas. Concretamente, como se muestra en la Figura 5.2, se definen seis etapas en el ciclo de vida de toda *botnet*. Aunque más adelante se describirán en mayor detalle, a continuación se introduce una pequeña descripción de las mismas:

1. *Etapa de concepción*

   Esta es la primera etapa en el ciclo de vida de toda *botnet*. La motivación para crear una *botnet* es esencial para su posterior diseño e implementación. Las principales características de diseño de la *botnet* se definen en esta etapa y en ellas influye enormemente el objetivo final para el que está ideada la *botnet*.

2. *Etapa de reclutamiento*

   Tras la creación de una *botnet* es preciso reclutar a los miembros de la misma. Por tanto, un *botmaster* ha de infectar equipos convencionales con el código malicioso desarrollado para comenzar a formar su *botnet*.

3. *Etapa de interacción*

   Esta etapa implica dos procesos diferentes. Primero, los *bots* infectados deben ser registrados en la *botnet* para que así comiencen a formar parte de su funcionamiento habitual. Segundo, debe existir un mecanismo que permita al *botmaster* comunicarse con sus *bots*. Estas comunicaciones se realizan por medio del canal C&C. El intercambio de información se compone de *órdenes* (*command*), que el *botmaster* envía a sus *bots*, y *operaciones de mantenimiento* (*control*), tales como actualizaciones del código de la *botnet*, recuento de miembros, etc.

4. *Etapa de marketing*

   Aunque la motivación más común para que un desarrollador diseñe e implemente una *botnet* es obtener un beneficio económico, existen otras razones posibles que incluyen el propio ego, la política, el reconocimiento de ciertos grupos sociales, etc. Sea cual sea la motivación, existe una etapa publicitaria (*marketing*) en la que el desarrollador muestra las ventajas y capacidades de su *botnet* en un foro específico, con objeto de ceder el uso total o parcial de la misma y obtener así un beneficio.

5. *Etapa de ejecución del ataque*

   En esta etapa, el *botmaster* ordena a los *bots* la ejecución de un ataque. Como se comentó anteriormente, una de las principales características de las *botnets* es el enorme número de *bots* de los que puede disponer para llevar a cabo actividades maliciosas. Por tanto, las *botnets* son armas especialmente diseñadas para realizar ataques que requieran un elevado número de participantes, como son: DDoS, fraude del *click* o *phishing*, entre otros.

6. *Etapa de ataque satisfactorio*

   El objetivo último de toda *botnet* es ejecutar satisfactoriamente el ataque o ataques para los que fue diseñada.

Nótese que cada etapa del ciclo de vida representa únicamente el inicio de la ejecución de un proceso específico. Claramente, si la *botnet* alcanza la etapa de *ataque satisfactorio*, el *botmaster* tratará de utilizarla de nuevo para los mismos u otros ataques (*etapa de concepción*). En ese mismo sentido, durante todo el funcionamiento de la *botnet* se intentarán reclutar miembros para que ésta aumente el número de sus activos (*etapa de reclutamiento*) y será necesario comunicarse con ellos (*etapa de interacción*). Por esta razón, cuando una etapa ha sido completada por una *botnet*, cualquier proceso dentro de esta etapa o de las anteriores puede ser reejecutado.

Como se muestra en la Figura 5.2, existen también varios mecanismos complementarios a las etapas del ciclo de vida de las *botnets*. Estos mecanismos se centran normalmente en la ocultación de la *botnet* (la localización de los *bots* y el *botmaster*, los procesos de comunicación, etc.). Algunos ejemplos de mecanismos de ocultación pueden ser: utilización de conexiones con múltiples saltos, *spoofing* de IP o redes *fast-flux*. Estos mecanismos se describen en la Sección 5.3.6.

Como se mencionó en la introducción, la comunidad investigadora ha realizado un gran esfuerzo con el objetivo de neutralizar las *botnets* y de evitar su construcción. Estos dos objetivos, sin embargo, requieren estrategias totalmente diferentes como se discutirá más adelante. En este punto es importante resaltar que cualquier aproximación dedicada a la inutilización de una *botnet* está realmente enfocada en la interrupción de la ejecución de un proceso específico de una de las etapas anteriores, o bien de una combinación de procesos de una o más etapas. En consecuencia, se puede deducir que una medida capaz de evitar la ejecución de cualquiera de estas etapas sería capaz de frustrar el funcionamiento general de la *botnet*. Por ejemplo, si una medida de defensa fuera capaz de impedir la infección de *bots* sería imposible reclutar miembros suficientes para llevar a cabo un ataque, o bien los efectos de éste serían muy limitados. En resumen, se puede concluir que ***la interrupción de tan sólo una etapa del ciclo de vida de la* botnet *impedirá la consecución del objetivo final de la misma.***

Obviamente, la mayoría de las *botnets* actuales implementan mecanismos complementarios de ocultación que reducen la probabilidad de ser detectadas. Estos mecanismos de ocultación se presentan en el ciclo de vida propuesto como un aspecto adicional a las etapas y no como una etapa en sí misma. Esta decisión ha sido tomada en coherencia con el criterio de definición de las etapas, ya que evitar la ejecución de los mecanismos de ocultación no implica necesariamente frustrar la consecución del objetivo de la *botnet*, sino que solamente incremente las probabilidades de que ésta sea detectada por un mecanismo de defensa. En conclusión, el ciclo de vida

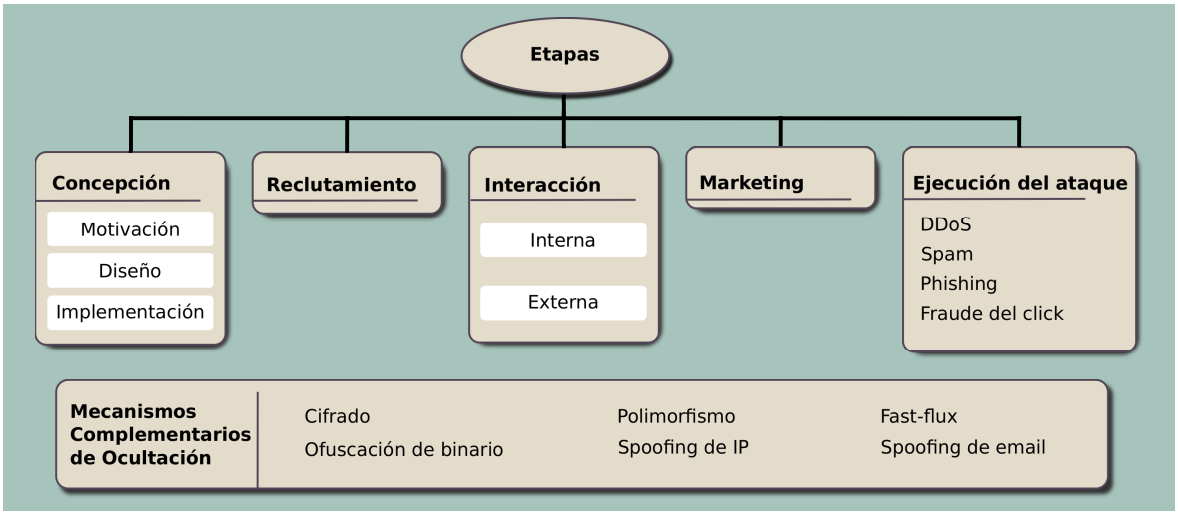


Figura 5.3: Taxonomía propuesta basada en el ciclo de vida de las *botnets*.

presentado es una cadena lineal de etapas en la cual si se rompe uno de los enlaces la *botnet* es inutilizada.

En las siguientes secciones se describen en mayor detalle las diferentes etapas y procesos del ciclo de vida de las *botnets*, tomando en consideración las diferentes alternativas de diseño e implementación que los desarrolladores pueden adoptar. El análisis de estas etapas ha permitido la creación de la taxonomía presentada en la Figura 5.3.

5.3.1. Concepción

La primera etapa del ciclo de vida es la etapa de *concepción* de la *botnet*. Es importante comprender tanto las razones que mueven a un desarrollador a crear una *botnet*, como las arquitecturas y diseños utilizados más frecuentemente.

La etapa de concepción se puede dividir en tres fases o procesos sucesivos (véase Figura 5.4): *motivación*, *diseño* e *implementación*.

Motivación

Un desarrollador en potencia necesita una razón que lo motive para crear una *botnet*. Las motivaciones que mueven a los desarrolladores de *malware* han sido clasificadas como: económicas, de entretenimiento, ego, causas personales, entrada en un grupo y *status* social (su acrónimo en inglés es MEECES) [76]. Sin embargo, de entre éstas, la principal motivación es la económica. El informe de Symantec

Figura 5.4: Procesos que componen la etapa de concepción de una *botnet*.

publicado en abril de 2010 [77], muestra un catálogo detallado con los precios de los servicios publicitados por una serie de *botnets*. Por ejemplo, el kit de la *botnet* Zeus cuesta alrededor de 700 dólares.

Sean cuales sean las razones que un desarrollador pueda albergar, los procesos que siguen a la motivación son el diseño e implementación de la *botnet* deseada. El desarrollador debe tomar algunos aspectos en consideración previos a estos procesos, especialmente los relativos a la infección de los *bots* y a las comunicaciones de la *botnet* (descritos en las siguientes secciones). Esto se debe a que en este punto se define la arquitectura de la *botnet*, característica que influirá de forma determinante en la operación de la misma.

Diseño

La arquitectura de una *botnet* puede ser *centralizada*, *distribuida* o *híbrida*. En un esquema *centralizado*, los *bots* contactan con un servidor C&C para recibir la información enviada por y hacia el *botmaster*. En general, se invierte muy poco tiempo en enviar un mensaje a todos los *bots* activos en la red. Ésta es una de las principales ventajas que aporta este esquema, unido a su sencillez de diseño e implementación. Por contra, su desventaja principal es el hecho de que el servidor C&C constituye un punto único de fallo. Si el servidor cae, la red completa es neutralizada ya que el *botmaster* no será capaz de enviar órdenes a sus *bots*. Algunos ejemplos relevantes de *botnets* centralizadas son: Eggdrop [78], Gt-Bot y Agobot [74] y Bobax [79].

A fin de evitar la limitación que acarrea el punto único de fallo de la arquitectura centralizada, algunas *botnets* adoptaron el algoritmo DGA (*Domain Generation*

Algorithm) para determinar en cada caso el servidor central al que los *bots* han de conectarse. Este es el caso de la conocida Conficker [80].

En una arquitectura *distribuida* todos los *bots* de la *botnet* actúan a la vez como servidores y como clientes. Con esta filosofía no existe un punto único de fallo. Este tipo de *botnets*, por tanto, es más robusto a los ataques de los agentes de defensa que el tipo de *botnet* basado en una arquitectura centralizada. Sin embargo, el tiempo que un mensaje requiere para alcanzar todos los *bots* activos de la red es muy superior al necesario en el caso de *botnets* centralizadas. Existen multitud de *botnets* con esta arquitectura, como por ejemplo: Spybot [67], Storm [81], Nugache [82] y Phatbot [83].

Finalmente, las *botnets* con arquitectura *híbrida* combinan las ventajas de las dos arquitecturas previas. En este caso, existe al menos una red distribuida y cada una de ellas presenta uno o varios servidores centralizados. La desconexión de uno de estos servidores sólo implica, en el peor de los casos, la caída de una de las redes distribuidas, permitiendo que el resto de la red continúe con su funcionamiento normal. Algunos ejemplos relevantes de este tipo de *botnets* son: Torpig [84], Waledac [85] y un nuevo diseño de *botnet* propuesto por Wang et al. en [86]. Las *botnets* híbridas más recientes y sofisticadas son Alureon/TDL-4 [87] y Zeus-P2P [88]. Alureon/TDL-4 fue la segunda *botnet* más activa en la primera mitad del año 2010, de acuerdo con un estudio realizado por Microsoft. Por otro lado, Zeus P2P, también conocida como Murofet v2.0, es un avance de la clásica Zeus diseñada para robar información personal que es enviada periódicamente a múltiples servidores HTTP centralizados.

Aparte de la arquitectura deseada, una decisión importante de diseño es si el desarrollador ha de implementar un nuevo protocolo para la arquitectura o si utilizará un protocolo y una red ya existente. Este es el caso de un tipo de *botnet* denominado *botnet parásita*. Esta *botnet* actúa como un parásito en el sentido de que su desarrollo, propagación y operación se sustenta en la existencia de toda una infraestructura de comunicaciones ya disponible. Las *botnet* parásitas conocidas utilizan una arquitectura distribuida. En este caso, en sus comunicaciones utilizan una red P2P legítima para enviar los mensajes C&C a través de ella. Estas *botnets* son especialmente difíciles de detectar, ya que sus mensajes C&C quedan ocultos entre el elevado tráfico de la red legítima que utilizan para comunicarse. En la actualidad se han reportado dos *botnet* parásitas: Storm [81] y una evolución concreta de Alureon [87]; ambas se comunican a través de Kad (una implementación de Kademlia, el algoritmo de tablas *hash* distribuidas, para la red eDonkey).

Implementación

Una vez que la *botnet* ha sido convenientemente concebida y diseñada, el último proceso de esta etapa se refiere a la implementación concreta de la *botnet*. Esta tarea no presenta ninguna característica particular y puede ser realizada siguiendo el proceso tradicional de desarrollo de software.

5.3.2. Reclutamiento

Una vez diseñado e implementado, el software de la *botnet* debe ser puesto en marcha en un entorno real. Para este propósito es necesario reclutar *bots*, miembros para la *botnet*. Al *botmaster* le interesa reclutar tantos *bots* como le sea posible, ya que esto aumenta notablemente el potencial de su *botnet*. Nótese que esta cuestión no es particular al caso de las *botnets*, sino que es común a muchas técnicas de ataques en red. El reclutamiento de miembros, más conocido como *infección*, ha sido ampliamente estudiado en la literatura específica de seguridad en redes.

En este sentido, no se conocen técnicas especialmente diseñadas para el caso del reclutamiento de *bots*, sino aquellas utilizadas en la propagación de *malware* en general. Para incrementar las capacidades de reclutamiento de una *botnet* es común introducir en el código malicioso multitud de *exploits* existentes y algunos *bugs* aún sin reportar (*exploits* de día cero). Una prueba de la escala del problema de la variedad de vectores de infección existentes es que en los tres últimos años se publicaron más de 14.000 vulnerabilidades nuevas [89]. Como ejemplo de una *botnet* con un método de reclutamiento particular, se puede citar a la *botnet* Stuxnet [90], que basa la infección en una vulnerabilidad de los sistemas SCADA (*Supervisory Control And Data Acquisition*) de las plantas nucleares. Como ejemplo de la importancia de esta fase, el éxito de la *botnet* Agobot [91] se ha debido fundamentalmente a su versatilidad en el reclutamiento, al contener más de 10 exploits diferentes. Por este motivo, muchas *botnets* posteriores han basado su código en ella, *p.ej.*, Phatbot [83].

Según Provos et al. [92], en los últimos años el reclutamiento se está orientando principalmente a la explotación de vulnerabilidades de servidores remotos, que son identificados por medio de escaneos en busca de redes vulnerables en Internet. Las técnicas de propagación automática como las utilizadas por Conficker [80] son un ejemplo de estos escaneos. Otros métodos de reclutamiento incluyen el envío de millones de correos electrónicos con enlaces que apuntan a páginas que se aprovechan de una vulnerabilidad del navegador para descargar y ejecutar un código malicioso (ataques *drive-by-download*) en el equipo destino [93]. De este modo, una víctima puede ser infectada solamente visitando una página web [94]. Actualmente, la propagación de *malware* por medio de redes sociales es también un mecanismo de infección en crecimiento [95] [96].

5.3.3. Interacción

En esta etapa se agrupan todas las interacciones que la *botnet* realiza durante su operación normal, *p.ej.*, las órdenes enviadas por el *botmaster*, los mensajes intercambiados por los *bots* o las comunicaciones con servicios externos a la *botnet* tanto del *botmaster* como de los *bots*.

Figura 5.5: Procesos que componen la etapa de interacción de una *botnet*.

Una de las principales diferencias entre las *botnets* y cualquier otro tipo de *malware* radica en que el *botmaster* puede enviar órdenes a los equipos infectados sin necesidad de controlar una a una estas máquinas por escritorio remoto. De esta forma, los *bots* pueden actuar de forma coordinada siguiendo las instrucciones del *botmaster*. Para realizar toda esta coordinación se precisa un canal de comunicaciones, denominado canal C&C. Los mensajes C&C presentan una relevancia especial, ya que portan información crucial de la *botnet* y, por esta razón, muchos de los trabajos de investigación dedicados a detectar *botnets* se basan en ellos. Por este motivo, la etapa de interacción de la *botnet* presenta habitualmente un interés especial para la comunidad investigadora.

En la Figura 5.5 se muestran los procesos que componen esta tercera etapa. Estos procesos pueden clasificarse como: *interacciones internas* e *interacciones externas*.

Interacciones internas

Las interacciones internas son aquéllas que se producen entre miembros de la *botnet*, es decir, desde el *botmaster* a los *bots* o viceversa, o solamente entre los *bots*. En estas interacciones se pueden encontrar dos procesos diferentes: *registro* y *comunicaciones C&C*.

Proceso de registro

El registro es el proceso por medio del cual un equipo comprometido pasa a formar parte de una *botnet*. Este proceso se puede identificar también en el funcionamiento de otros tipos de redes, como pueden ser las redes P2P, en las que a este proceso se le denomina *bootstrap*. En el campo que nos ocupa existen dos tipos de registros: *estáticos* y *dinámicos*.

En el registro estático, toda la información necesaria para llegar a ser parte de la *botnet* está contenida en el código que infectó al *bot* (en inglés, *hardcoded*). Así, generalmente, la dirección IP del servidor C&C se facilita (con algún tipo de ofuscación o cifrado) en el código del *bot*. GT-Bot, Agobot y SDbot [74] son algunos ejemplos de este tipo de registro. En el caso de las *botnets* distribuidas o híbridas, el registro estático es más complejo. Un nuevo equipo infectado puede utilizar una lista de direcciones IP contenidas en el código del *bot* que apuntan a miembros de la *botnet*, a fin de enviarles solicitudes de registro. Waledac [85] es un ejemplo de esto. Otro ejemplo de una *botnet* distribuida que realiza un proceso de registro estático es Sinit [97], donde cada nuevo *bot* prueba un número de direcciones IP aleatorias dentro de un rango hasta que encuentra un miembro de la *botnet*, el cual, posteriormente, le ayuda en el proceso de registro.

En el registro dinámico los *bots* han de solicitar explícitamente la información necesaria para formar parte de la *botnet*. Esto suele llevarse a cabo por medio de una tercera parte no comprometida. Un ejemplo de este tipo de registro es el utilizado por Phatbot [83], en el que los nuevos *bots* consultan a los servidores de caché de Gnutella para descargar una lista de nodos que pertenecen a la red. Una vez que poseen esta lista, se consultan los nodos de ella hasta que se encuentra un miembro de la *botnet*. Tras esto, se realiza el proceso de registro con el miembro hallado.

Comunicaciones C&C

El grueso de las interacciones de una *botnet* se producen tras el proceso de registro. Estas interacciones son las comunicaciones C&C y pueden ser agrupadas en función de las siguientes tres características:

- *Tipo de mensajes*

  Los mensajes C&C pueden clasificarse como *comandos* (órdenes) o de *control*. Los comandos son enviados por el *botmaster* y tienen como objetivo indicar a los *bots* la acción concreta a realizar. Por ejemplo, la ejecución de un ataque DDoS a una hora determinada y contra un sitio web específico. Por otro lado, por medio de los mensajes de control se pretende recopilar información acerca de la *botnet*, por ejemplo, acerca del número de *bots* activos y las direcciones IP asociadas a éstos.

- *Dirección de la información*

  Los mensajes C&C también pueden clasificarse como *pull* o *push* [98]. Los *bots* pueden solicitar información activamente a los servidores C&C por medio del envío de mensajes *pull*, que viajan desde los *bots* a los servidores C&C, o recibir información de forma pasiva sin enviar previamente ningún mensaje de solicitud a través de los mensajes *push*, que viajan desde los servidores C&C hacia los *bots*.

- *Protocolo de comunicación*

  Otra característica importante que permite clasificar los mensajes C&C es el protocolo o protocolos implicados en las comunicaciones. En este sentido existen dos opciones fundamentales:

  - Utilizar un protocolo específicamente diseñado para las comunicaciones de la *botnet*, es decir, un *protocolo propietario*.
  - Aprovechar una infraestructura ya existente por medio de un *protocolo estándar* como puede ser: IRC, HTTP o algunos protocolos P2P. Las *botnets* que se aprovechan de estas redes y de sus protocolos son las ya mencionadas *botnets* parásitas.

Interacciones externas

Las interacciones externas son aquéllas que implican comunicaciones desde miembros de la *botnet* hacia servicios externos a ésta. Estas comunicaciones corresponden generalmente a accesos a servicios ofrecidos en Internet. Los principales servicios externos utilizados son los siguientes:

- *Servicio de DNS*

  En una *botnet* centralizada es común que los *bots* envíen consultas DNS para resolver la dirección IP del servidor C&C. Este es el caso de la mayoría de las *botnets* que utilizan IRC para comunicarse, como Agobot, GT-Bot o SDBot [91], entre otras. También existen *botnets* que utilizan el protocolo HTTP, como Bobax [79], que también precisan la utilización del servicio DNS.

- *Servicios de redes P2P*

  Una tendencia común en el diseño de *botnets* es el uso de redes P2P como capa intermedia para ocultar comunicaciones C&C. En lugar de almacenar en el código del *bot* las direcciones IP de los servidores C&C, los *botmasters* guardan los nombres de un recurso que debe ser descargado desde una red de compartición de archivos mediante P2P. Este recurso contiene una serie de instrucciones del *botmaster*, probablemente una orden para un ataque específico,

o simplemente las especificación de un nuevo recurso que debe ser descargado en un momento temporal concreto para actualizar el código del *bot*. Por ejemplo, Trojan.Peacomm [81] explora la red Overnet en busca de claves generadas por un algoritmo propio. Estas claves apuntan a archivos que contienen las URLs desde las que los *bots* han de descargar sus actualizaciones. En estas *botnets*, derivado de estas interacciones, se genera un volumen de tráfico considerable dentro de la red P2P.

- *Servicios de monitorización de* botnets

  Un *botmaster* consulta este tipo de servicios con el objetivo de obtener información útil acerca de si su *botnet* ha sido detectada o no. Por ejemplo, consultar la disponibilidad de los dominios utilizados por un servidor C&C puede comprobarse a través de un servidor Whois o un servicio de DNSBL (*DNS-based Block List*), en el que se facilita una lista con los dominios bloqueados (*blacklisted*). Esta última información es utilizada en algunos sistemas propuestos para detección de *bots* como por ejemplo en [99], en la que mediante estas consultas se detectan servidores C&C a través de los que se identifica un gran porcentaje de los *bots* de una *botnet*.

5.3.4. Marketing

Una vez que la *botnet* ya ha sido creada y es funcional se alcanza la etapa de marketing. En este punto, el *botmaster* necesita un incentivo para usar su *botnet*. Aunque existen muchas razones posibles – entretenimiento, ego, reconocimiento, etc. – la más común es percibir un beneficio económico.

El beneficio económico es obtenido frecuentemente de dos formas: (*i*) vendiendo el código de la *botnet* o (*ii*) alquilando los servicios de ésta. En ambos casos es preciso llevar a cabo un proceso de marketing a través del que el *botmaster* muestra las capacidades de su *botnet* a los usuarios interesados en alquilar ciertos servicios o en adquirirla en su totalidad.

Aunque los desarrolladores de *botnets* pueden vender el código fuente de éstas, la aproximación más frecuente es distribuir un kit de desarrollo, DIY. Como se mencionó en la Sección 5.2, con un kit de este tipo incluso los usuarios con pocos conocimientos en desarrollo de software son capaces de generar los binarios de los *bots*. Un ejemplo de un kit DIY es Turkojan 4, que está disponible para su descarga gratuita en Internet [100]. Otro ejemplo puede ser el kit de desarrollo de la *botnet* Zeus, que se publicita en los foros clandestinos por algo más de 700 dólares, mientras que el código fuente de una *botnet* menos sofisticada como es NETTICK se vende por unos 1.200 dólares [101].

La opción más común en el marketing de *botnets* es el alquiler de los servicios de éstas. A continuación, se presenta una lista con algunos de los servicios más

Tabla 5.2: Resumen de precios de los servicios más comunes ofertados por una *botnet*

	Precio estimado
DDoS	50$-miles$/día
Direcciones e-*mail*	20-100/millón de correos
Spam	150-200/millón de envíos
Cuentas de usuario	7-15/cuenta
Redes *fast-flux*	1.000-2.000/mes
Fraude del *click*	300$/mes

comunes y sus precios en el mercado clandestino, realizada por los laboratorios de Karspersky [102] (ver Tabla 5.2):

- *Ataques DDoS*

  Este tipo de ataques puede ser contratado por un precio que oscila desde 50 dólares hasta varios miles de dólares al día. El precio del ataque principalmente depende del número de *bots* que deben atacar y, por tanto, de la intensidad del ataque.

- *Direcciones e-mail*

  Las empresas de publicidad pueden estar muy interesadas en conseguir correos electrónicos a los que poder enviar sus anuncios. Obtener una lista de direcciones de correo de alrededor de un millón de direcciones oscila entre 20 y 100 dólares.

- *Envío de* spam

  El envío de mensajes publicitarios no solicitados por el receptor (*spam*) a un millón de direcciones puede ser contratado por precio que va desde 150 a 200 dólares.

- *Cuentas de usuario*

  Se pueden obtener, de los equipos infectados por la *botnet*, cuentas de usuarios de servicios on-line, *p.ej.*, por medio de *keyloggers*. Cada una de estas cuentas y su respectiva contraseña cuesta entre 7 y 15 dólares.

- *Redes* fast-flux

  Los cibercriminales, principalmente los que realizan *phishing*, pagan al mes a los propietarios de las *botnets* por alojar su página web en un servicio de *fast-flux* entre 1.000 y 2.000 dólares.

- *Falseo de los motores de búsqueda*

  A los *webmaster* les interesa que sus páginas web aparezcan como uno de los primeros resultados de los motores de búsqueda, para así tener más probabilidades de captar la atención del usuario. Estas posiciones pueden ser falseadas,

p.ej., introduciendo comentarios falsos con enlaces a la página cuya posición quiere ser mejorada. Esto puede llevarse a cabo por medio de una *botnet*, a un precio de 300 dólares al mes.

5.3.5. Ejecución del ataque

El objetivo final de una *botnet* es ejecutar de forma exitosa el ataque para el que fue concebida. La principal característica de estos ataques es la intensidad con la que son ejecutados, dado el elevado número de atacantes que toman parte en ellos. Esto no implica que una *botnet* no pueda ejecutar ataques ideados para un número limitado de participantes, sino que están especialmente diseñadas para aquéllos que, por contra, requieran un número elevado.

Los ataques principales ejecutados por *botnets* son los siguientes:

- *Ataques de denegación de servicio distribuidos (DDoS)*

  Los ataques DoS se definen como intentos de prevenir el uso legítimo de un servicio o simplemente de reducir su disponibilidad. Los ataques DDoS son un caso particular de éstos, en el que operan simultáneamente una enorme cantidad de atacantes (contra una o más víctimas) para conseguir este objetivo [103]. Las *botnets* son especialmente apropiadas para lanzar este tipo de ataques al poseer un gran número de miembros que se pueden coordinar entre ellos.

  Las medidas de defensa basadas en el filtrado de las direcciones IP origen no son válidas porque los *bots* están distribuidos geográficamente y no comparten prefijos IP. Además, estas redes son tan heterogéneas que emulan el comportamiento de miles de clientes legítimos, haciendo extremadamente complejo construir esquemas de defensa frente a ataques DDoS ejecutados por *botnets*. Algunos ejemplos de *botnets* utilizadas para ejecutar DDoS son Spybot y Agobot [91].

- *Spamming*

  Un correo electrónico *spam* contiene cierta información confeccionada para ser enviada a un enorme número de destinos, lo deseen éstos o no [104]. El uso de *botnets* para este fin está muy extendido. Esto se debe a que incrementan considerablemente el número de correos que pueden enviarse por segundo, al disponer de una capacidad de subida suma de las capacidades de subida de cada uno de sus miembros. Además, se pueden utilizar para enviar correos desde los servidores SMTP asociados a los usuarios de los *bots* infectados y, por tanto, la recepción de estos mensajes no puede ser filtrada simplemente inspeccionando el servidor de envío. Bobax es un ejemplo de *botnet* utilizada para este propósito [79].

- *Phishing*

  Esta actividad fraudulenta puede definirse como la creación de una réplica de una página web existente o de otro recurso web, con el objetivo de engañar a un usuario y conseguir que éste envíe información personal, financiera o contraseñas y/o datos privados [105]. Estas réplicas implican un coste elevado en desarrollo y mantenimiento y los encargados de ellas (*phishers*) invierten un gran esfuerzo en ocultarlas. Para este propósito, los *phishers* últimamente utilizan una técnica muy común denominada redes *fast-flux*, que será descrita en la Sección 5.3.6.

- *Robo de datos*

  Este ataque tiene como objetivo robar información sensible a los usuarios de los equipos infectados por la *botnet* que lanza el ataque. Este robo se realiza utilizando técnicas comunes a otros *malware* como la inspección de archivos en búsqueda de palabras clave, el robo de *cookies*, el uso de *keyloggers*, etc. Zeus [106] es un conocido ejemplo de *botnet* que utiliza este tipo de ataques; en concreto es capaz de, entre otras acciones, robar *cookies* guardadas por Paypal y enviarlas al *botmaster*.

- *Fraude del* click

  Este ataque consiste en inducir a usuarios, mediante el engaño, para que realicen *clicks* en anuncios online o que visiten ciertas páginas y, de esta forma, incrementar las ganancias de terceras partes relacionadas con los anuncios publicitarios [107]. El uso de las *botnets* hace posible simular el comportamiento de millones de usuarios legítimos siendo ideales para este tipo de ataques. La *botnet* Clickbot.A es un ejemplo del uso del ataque de fraude del *click* [108].

5.3.6. Mecanismos complementarios de ocultación

Como se muestra en las Figuras 5.2 y 5.3, los mecanismos complementarios de ocultación se consideran parte del ciclo de vida de las *botnets* propuesto. Estos mecanismos se utilizan para ocultar la *botnet* y hacer más difícil la detección de sus componentes (*bots*, *botmaster*, canales C&C). Desafortunadamente, incluso si un mecanismo de ocultación fuera detectado y neutralizado, la *botnet* no sería neutralizada. La premisa básica en la que nos basamos para diseñar el presente ciclo de vida es que si una etapa del mismo es interrumpida la operación completa de la *botnet* es frustrada. Así que, por coherencia con esta premisa de diseño, estos mecanismos no se consideran una etapa del mismo sino complementarios a aquéllas.

Existen multitud de técnicas de ocultación que pueden ser utilizadas en el campo de las *botnets*. En lo que sigue se presentan algunas de las más usuales:

- *Utilización de conexiones multi-salto*

  El atacante conecta con múltiples *proxies* localizados en diferentes países antes de conectarse con cualquier *bot*, generando incluso bucles en dichas conexiones. Como consecuencia de esta técnica, descubrir la dirección IP desde la que se establece realmente la conexión es mucho más complejo. Este esquema de ocultación se utiliza típicamente en la etapa de interacción, *p.ej.*, en las comunicaciones C&C del *botmaster* con los *bots*. La técnica de utilización de conexiones multi-salto también podría utilizarse en la etapa de marketing, *p.ej.*, antes de publicar un mensaje en un foro clandestino para anunciar los servicios de una *botnet*.

- *Cifrado*

  En las *botnets* actuales, los canales C&C utilizados en la etapa de interacción se cifran a fin de prevenir que un nodo intermedio pueda tanto interceptar y comprender los mensajes C&C, y también evitar que se puedan enviar mensajes falsos a los *bots*. El desarrollo de técnicas para detectar canales C&C llega a ser aún más complejo con el uso de técnicas de cifrado. SpamThru [109] y Zeus [106] utilizan canales C&C encriptados. Phatbot [83] utiliza encriptación en el protocolo P2P WASTE, protocolo propietario desarrollado para realizar las comunicaciones C&C de forma segura en redes P2P.

- *Ofuscación de binarios*

  Los desarrolladores utilizan técnicas de ofuscación de los archivos binarios que utilizan para infectar a un equipo con el objetivo de ocultar el código fuente de este *bot* [110]. Esto dificulta el análisis del código de los *bots* por medio de técnicas de ingeniería inversa y, por tanto, el análisis del comportamiento de la *botnet* en general. La *botnet* Conficker [80] utiliza esta técnica de una forma avanzada. Realiza multitud de llamadas indirectas y saltos a distintas zonas del código y divide sus funciones en secciones de código muy pequeñas, complicando enormemente su posterior análisis por parte de los agentes de seguridad.

- *Polimorfismo*

  Esta técnica consiste en crear diferentes versiones del mismo código fuente de un programa, que cambian la huella del binario sin alterar las funcionalidades de éste. El uso de polimorfismo hace casi inútil el uso de técnicas de detección basadas en firmas utilizadas por la mayor parte de los antivirus. El polimorfismo mejora la capacidad de infección de la *botnet* durante la etapa de reclutamiento al disminuir la probabilidad de ser detectado como una amenaza. Phatbot [83] y Zeus [106] son ejemplos de *botnets* que utilizan esta técnica.

Tabla 5.3: Mecanismos complementarios de ocultación presentes en cada etapa

	Concepción	**Reclutamiento**	**Interacción**	**Marketing**	**Ataque**
Conexiones multi-salto			x		
Cifrado			x		
Ofuscación de binario	x	x			
Polimorfismo		x			
Spoofing de correo		x		x	x
Spoofing de IP				x	x
Fast-flux		x	x	x	x

- *Falseamiento de direcciones IP (IP* Spoofing*)*

  Se define como un falseo de la identidad del atacante por medio del envío de paquetes con una dirección IP origen falsa. Esta técnica se utiliza ampliamente en los ataques DDoS para así evitar filtros basados en IP y para ocultar la identidad del atacante.

- *Spoofing de correo electrónico*

  De la misma forma que el *spoofing* de IP, el *spoofing* de correo electrónico consiste en enviar correos en los que se falsea la dirección de correo origen u otros campos de la cabecera. Esto se utiliza comúnmente en ataques de *phishing*, como los perpetrados por la *botnet* Bobax [79].

- *Redes* fast-flux

  En estas redes existe un elevado número de *proxies* que redirigen las solicitudes de los usuarios al servidor objetivo, por ejemplo un servidor central C&C o un servidor web para realizar un *phishing*. Estos *proxies* cambian muy frecuentemente utilizando entradas DNS con un reducido TTL (*Time To Live*) para el dominio *flux*, lo que implica que seguir las comunicaciones sea extremadamente complejo. Para el propietario de una red *fast-flux* es importante disponer del mayor número posible de *proxies* y también es deseable que las localizaciones geográficas de éstos sean heterogéneas (de diferente redes). Es innegable que los *bots* cumplen ampliamente estas dos premisas y, por este motivo, las *botnets* son una herramienta ideal para implementar redes *fast-flux*. Por ejemplo, la *botnet* Storm utiliza este mecanismo para ocultar las comunicaciones implicadas en la actualización del código de sus *bots* [111].

En la Tabla 5.3 se listan todos los mecanismos de ocultación presentados anteriormente y se indica la etapa o etapas en las que estos son utilizados más frecuentemente.

5.4. Revisión de la investigación sobre *botnets* en base a su ciclo de vida

El propósito principal para llevar a cabo esta revisión es aportar orden y facilitar la comprensión de la enorme cantidad de trabajos de investigación relacionados con el tema de las *botnets*. Con este fin resultará de gran utilidad la taxonomía propuesta. Se realiza a continuación una selección de los artículos más relevantes directamente relacionados con las *botnets* y posteriormente serán clasificados dentro de las diferentes etapas del ciclo de vida de éstas. Con este trabajo también se pretende facilitar el diseño de defensas efectivas frente a *botnets*.

Es interesante destacar que la mayoría de las contribuciones que se mostrarán y clasificarán a continuación están enfocadas a resolver el problema de las *botnet* centrándose en una etapa de las presentadas anteriormente en el ciclo de vida de una *botnet*. Esto confirma nuestra hipótesis de que la neutralización de una sola etapa realmente inutiliza la *botnet*.

5.4.1. Concepción

La mayoría de los artículos sobre *botnets* están realmente enfocados en el estudio de algunas *botnets* en particular o sugieren mecanismos de defensa frente a éstas. En ellos se destaca que un buen punto de partida en la construcción de un esquema de defensa efectivo contra las *botnets* podría ser desmotivar a los potenciales desarrolladores de iniciar la tarea de diseño de una nueva *botnet*. Una estrategia inicial para alcanzar este objetivo es implementar medidas legales específicas con el objetivo de desmotivar a estos desarrolladores potenciales (véase la Sección 3.1 de [112]). Otra aproximación posible para dificultar la implementación de *botnets* es diseñar y poner en funcionamiento redes de comunicaciones con protocolos específicos que eviten o dificulten las comunicaciones de *botnets* a través de ellas. Sin embargo, hasta la fecha, este campo aún no ha sido explorado por la comunidad investigadora.

La mayoría de la investigación se ha centrado en intentar comprender el comportamiento y la arquitectura de las *botnets* y cómo éstas se diseñan e implementan. Estos estudios se utilizan usualmente como un paso previo al descubrimiento de mecanismos eficaces de detección. Así, existen multitud de artículos de *botnets* enmarcados en la etapa de concepción que pueden ser clasificados en tres grupos: *estrategias de infiltración en* botnets, *estudio de* botnets y *nuevos diseños de* botnets.

Estrategias de infiltración en *botnets*

En este grupo de contribuciones la estrategia de los investigadores pasa por formar parte de la *botnet* para estudiar su funcionamiento desde dentro. Generalmente, la

infiltración en una *botnet* no es un fin en sí mismo, sino un medio para estudiar una *botnet* concreta. Aunque esta estrategia presenta dudas legales, ya que implica un ataque a un sistema, también es un método muy efectivo para aprender acerca del comportamiento de la *botnet* estudiada. El proceso de infiltración se lleva a cabo siendo infectado con el código de la *botnet* y, por tanto, convirtiéndose en un *bot* más de la red, la aproximación más común, o convirtiéndose en un servidor C&C, tomando así el control de la totalidad o de parte de la *botnet*.

Freiling et al. [113] describen un método general para infiltrarse como *bot* en una *botnet* IRC. Primero, se obtiene el *malware* utilizado por la *botnet*, utilizando *honeypots* o *Mwcollect*. El último es un programa, similar a *Honeyd* [114], desarrollado para capturar *malware* en entornos no nativos. Posteriormente, el *malware* es analizado para extraer información útil que permita la infiltración dentro de la *botnet* correspondiente, como servidor C&C, alias de un *bot* existente, clave de acceso si es necesaria, etc. Cremonini et al. [115] presentan Dorothy, un entorno libre para monitorizar la actividad de una *botnet* en la que previamente es preciso infiltrarse. En un caso de estudio que llevan a cabo estos autores, se infiltran y monitorizan una *botnet* llamada Siwa. Derivada de esta infiltración y utilizando Dorothy son capaces de recoger información de Siwa relativa a la estructura funcional, distribución geográfica, mecanismos de comunicación, lenguaje de comandos y operaciones.

El mecanismo que algunas *botnets* como Conficker, Kraken o Torpig utilizan para recibir órdenes del *botmaster*, ofrece una oportunidad única para infiltrarse en ellas como servidor C&C. Estas *botnets* utilizan un cambio constante de dominio (*domain* flux), de modo que cada *bot* periódicamente (e independientemente) genera una lista de dominios con los que contactar. Después, el *bot* intenta conectarse a ellos uno tras otro. El primer equipo que envíe una respuesta identificándose a sí mismo como un servidor C&C válido es considerado legítimo, hasta que el siguiente periodo de generación de dominios comience. Para infiltrarse en estas *botnets*, los investigadores proponen metodologías que les permiten predecir el conjunto de nombres de dominio con los que los *bots* se pondrán en contacto. Seguidamente, este conjunto de nombres de dominio es registrado a fin de hacerse pasar como servidores C&C. Este es el caso de los trabajos para Conficker [116], para Kraken [117], y para Torpig [118].

Estudio de *botnets*

Los investigadores han llevado a cabo el estudio de *botnets* en dos niveles: a *nivel de* bot y a *nivel de red*. El estudio a *nivel de* bot consiste en analizar las acciones que realiza un *bot*. El principal objetivo de éste es extraer toda la información posible del código de los *bots*, como puede ser: el protocolo de comunicación utilizado, los ataques que se pueden ejecutar, el cifrado de las comunicaciones, el algoritmo de generación de dominios, etc. Por otro lado, los estudios a *nivel de red* pretenden fundamentalmente

descubrir las características generales de una *botnet*, como pueden ser: el número de *bots* activos, la tasa de entrada y salida de la red (*churn*), la actividad de la *botnet* en función del tiempo, etc.

Un estudio detallado sobre una *botnet* es de una gran ayuda a la hora de construir defensas frente a ésta. Por esta razón, una gran cantidad de artículos han estudiado *botnets* específicas utilizando estrategias diferentes. Estos trabajos se clasifican en tres tipos principales: (*i*) herramientas para facilitar el estudio de *botnets*, (*ii*) estudios de una *botnet* en concreto y (*iii*) estudios de características de las *botnets* en general. Seguidamente se desarrolla cada uno de ellos.

Herramientas para facilitar el estudio de botnets

El uso de ingeniería inversa para realizar un estudio completo de un *malware* es una tarea ardua. Para facilitar esto, existen algunas propuestas en la línea de automatización de ingeniería inversa. Una de éstas, que ha sido directamente aplicada al caso de las *botnets*, es [119], en la cual Caballero et al. proponen el uso de un protocolo de ingeniería inversa automática para reproducir los mensajes C&C de una *botnet*. Esto puede ser también utilizado con posterioridad para infiltrarse en la *botnet* estudiada.

Existen también otras posibilidades diferentes a la ingeniería inversa del código de los *bots* para analizar *botnets*. Por ejemplo, Jackson et al. [120] desarrollaron un sistema llamado SLINGbot que permite a los investigadores simular tráfico de *botnets* actuales y futuras, caracterizarlo y, de aquí, derivar técnicas de defensa efectivas. Otro ejemplo puede encontrarse en [121], en el que los autores proponen una aproximación para inferir máquinas de estados finitos de protocolos de red en un entorno realista y aplicarlas al análisis de los protocolos C&C de *botnets*.

Estudios de una botnet *en concreto*

Se ha realizado un gran esfuerzo por parte de la comunidad investigadora en la línea del estudio de una *botnet* en particular. Tomando una aproximación cronológica y en función de su arquitectura de comunicación, en la Figura 5.6 y a continuación se destacan las *botnets* más relevantes denunciadas y estudiadas por investigadores. Eggdrop [78] es la primera *botnet* IRC conocida y fue desarrollada originalmente por Robey Pointer en 1993 y actualmente está disponible para su descarga libre. Tras esto, aparecieron diferentes *botnets* con arquitecturas similares, como GT-Bot [91] [122] en 1998. Esta *botnet* fue desarrollada para ejecutar una instancia del cliente de *chat* mIRC junto a una serie de *scripts* y algunos binarios adicionales. En paralelo a esto se ejecutaba un programa para hacer invisible la ventana del cliente mIRC y así evitar que el usuario legítimo del equipo se percatara de la actividad del *bot*. En 2002, Agobot [91] [122] fue el primer *bot* en hacer uso de una variación propia del protocolo IRC para sus mensajes C&C. En el mismo año apareció SDBot [91] [122], cuyo funcionamiento consistía básicamente en una implementación compacta de

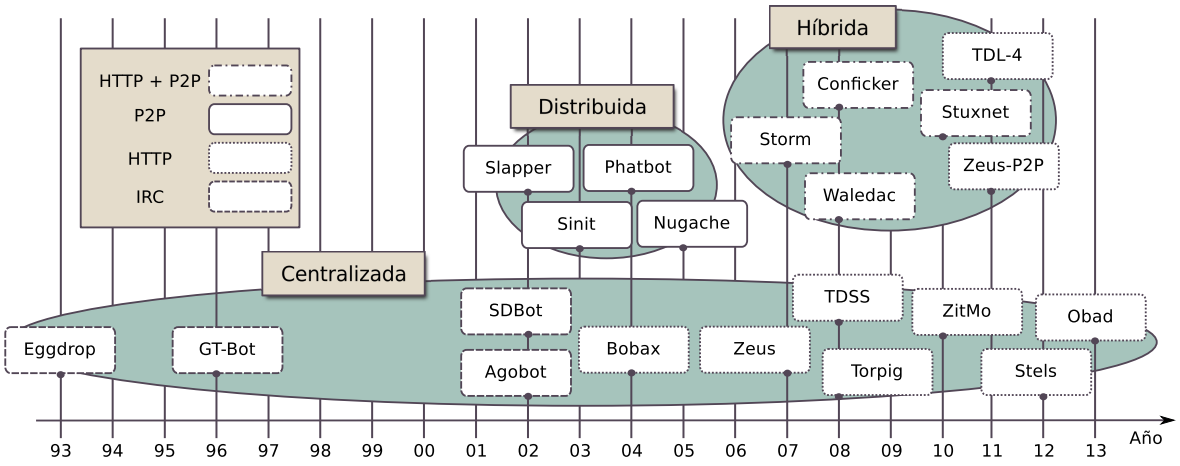


Figura 5.6: Evolución temporal de las *botnets* conocidas hasta la fecha, organizadas por su arquitectura de comunicación.

IRC con sus propias funciones. En 2003 apareció una versión avanzada de SDBot denominada SpyBot [91] [74]. Su código era eficiente pero no exhibía una gran modularidad y variedad de funcionalidades.

Una evolución natural de las *botnets* originales que se comunicaban a través de IRC son las basadas en el protocolo HTTP. Una de las ventajas principales de las *botnets* HTTP es que incluso mejoran la facilidad de desarrollo y puesta en funcionamiento de las *botnets* IRC. Como ejemplo, el tráfico de salida del protocolo HTTP siempre está permitido por los cortafuegos y casi cualquier equipo comprometido sería capaz de comunicarse con un servidor HTTP C&C. Entre este tipo de *botnets* se encuentra Bobax, que apareció en 2004, Zeus en 2007 (especialmente activa en 2010) y Torpig y TDSS en 2008. Los *bots* de Bobax se utilizan como nodos de retransmisión de correos electrónicos y, adicionalmente, intentan infectar otros equipos explotando ciertas vulnerabilidades o enviando correos *spam* [79]. Zeus [106] y Torpig [118] se diseñaron para robar información privada, principalmente de cuentas bancarias y utilizan técnicas de *phishing* para conseguir su objetivo. Torpig utiliza la técnica *domain* flux (cambio constante de nombres de dominio) para dificultar la tarea de detección y bloqueo de su servidor C&C. La *botnet* TDSS [123] se basa en un programa capaz de ocultar la presencia de otros *malware* en el sistema y otorgarles privilegios en el sistema infectado. Por defecto TDSS sólo implementa el ataque de fraude del *click* para, por ejemplo, conseguir dinero manipulando las estadísticas de tráfico de sitios web. En esencia, TDSS es un kit de desarrollo de *botnets* que permite incluir los ataques que los clientes que la compran necesiten y que ha sido constantemente actualizada. La segunda versión de este *malware*, TDL-2, apareció a principios de 2009 y a ésta le siguió la tercera versión, TDL-3, que apareció en agosto de 2009, todas estas versiones se comunicaban por medio del protocolo HTTP. Otro ejemplo de *botnet* HTTP es Twetbot [124], que utiliza la red social Twitter para controlar sus *bots*.

Una nueva tendencia en el cibercrimen es utilizar sistemas móviles como miembros de sus *botnets*. Este es el caso de la ingente cantidad de nuevas *botnets* centralizadas que utilizan el protocolo HTTP y que principalmente funcionan para el sistema operativo Android (un 93,94% de las amenazas pertenecen a esta plataforma [125]). Existen también evoluciones de las *botnets* convencionales con el objetivo de que éstas puedan funcionar en las plataformas móviles. Este es el caso de ZitMo [126] que vio la luz por primera vez en septiembre de 2010 y que es la evolución de la *botnet* Zeus para la plataforma Android. Stels [127] apareció en 2012 y tiene como objetivo principal el beneficio económico de los *botmasters* a través del envío de mensajes de texto de pago o robando información del dispositivo. Los *bots* se comunican con los servidores C&C por medio del protocolo HTTP mediante objetos JSON encriptados. Adicionalmente, si los *bots* pierden conectividad con sus servidores C&C pueden acceder a ciertas cuentas de Twitter de las que obtener una nueva lista de servidores C&C. En 2013 apareció Obad [128], una de las *botnets* más sofisticadas conocidas hasta la fecha para la plataforma Android. Esta *botnet* se comunica de una forma muy similar a Stel, por medio de objetos JSON encriptados. De esta *botnet* es destacable la capacidad que le brinda a los *botmasters* para comunicarse de forma inmediata con sus *bots* por medio de mensajes de texto con palabras clave como: `key_con`, con la que se solicita al *bot* a conectarse al servidor C&C, `key_die`, para eliminar las tareas pendientes, o `key_url`, con la que se solicita al *bot* que cambie la dirección del servidor C&C por la facilitada en este mensaje.

Todas las *botnets* descritas anteriormente son *botnets* centralizadas. Como se mencionó en la Sección 5.3.1, el problema principal de las arquitecturas centralizadas es que presentan un punto único de fallo localizado en el servidor C&C. Debido a esta deficiencia, las *botnets* evolucionaron de arquitecturas centralizadas a descentralizadas o distribuidas. La primera *botnet* conocida que utilizó una infraestructura P2P para sus comunicaciones C&C fue Slapper [129] que apareció en 2002. Esta *botnet* es una variación del gusano Scalper del servidor HTTP de Apache, con la peculiaridad de que utiliza un protocolo P2P en sus comunicaciones C&C. Estas comunicaciones utilizan el protocolo UDP y el puerto 2002, probablemente en honor al año de su creación. Más adelante, en el año 2003, apareció una *botnet* llamada Sinit [97]. Esta *botnet* almacena una lista de nodos conocidos y realiza un procedimiento de registro (*bootstrap*) escogiendo direcciones IP de forma aleatoria hasta que encuentra un miembro de la *botnet*. Phatbot [83], que apareció en 2004, es un descendiente directo de Agobot y utiliza los servidores de caché de Gnutella para encontrar miembros de la *botnet* y así realizar el proceso de registro. Los mensajes que se envían por medio del protocolo P2P están encriptados. La *botnet* Nugache [82] apareció en 2005 y su proceso de registro se realiza gracias a una lista de miembros de la *botnet* almacenada en el código del *bot*. También se comunica por medio de un canal C&C encriptado.

La última evolución relativa a la arquitectura de comunicaciones es la arquitectura híbrida, una solución intermedia entre las arquitecturas centralizadas y

distribuidas. La primera *botnet* conocida con esta arquitectura fue Storm (también conocida como Peacomm) [81] [130] [82]. Apareció en 2007 y fue una de las *botnets* más influyentes hasta la fecha debido a su reclutamiento masivo y a la dificultad para eliminarla. Los *bots* de Storm utilizaban la red Overnet, como protocolo P2P descentralizado, para encontrar sus servidores C&C HTTP. Storm también fue una de las primeras *botnets* en utilizar redes *fast-flux* y su propósito principal era el envío de correos *spam*. Posteriormente, en 2008 aparecieron dos *botnets* muy relevantes: Waledac y Conficker. Waledac [131] [132] [85] utiliza HTTP y protocolos P2P para sus comunicaciones C&C y funciona de una forma similar a Storm. Conficker [80] [133] también implementa sus comunicaciones C&C sobre HTTP y protocolos P2P y utiliza cambios constantes de nombres de dominio para esconder los servidores C&C HTTP. En 2010 apareció una *botnet* híbrida, Stuxnet [90], capaz de reprogramar sistemas de control (SCADA) modificando el código de algunos controladores lógicos programables (PLC (*Programmable Logic Controller*)). Su objetivo principal resultó ser ejecutar ataques contra plantas nucleares de Irán. TDL-4 [87], es una *botnet* que evolucionó de TDSS y apareció en 2011. Esta *botnet* utiliza comunicaciones P2P y centralizadas por medio del protocolo HTTP. Un *bootkit* se instala en el sistema infectado permitiendo cargar el código binario del *bot* antes que el sistema operativo, pudiendo así evitar los sistemas antivirus. Una diferencia principal entre TDL-4 y otras *botnets* híbridas es su librería `kad.dll`, que permite comunicarse por medio de KAD, la red P2P distribuida del cliente eMule. Esta comunicación se lleva a cabo de una forma similar a la de Storm, pero añadiendo un nivel de encriptación elevado. Finalmente, también en 2011, Zeus evolucionó a una arquitectura híbrida (Zeus-P2P) [88]. En esta versión la *botnet* Zeus utiliza una lista de miembros conocidos de la *botnet* grabada en su código. Los miembros de la *botnet* se comunican a través de un protocolo P2P propietario para obtener las actualizaciones del código del *bot* y de los archivos de configuración. Después, el *bot* se conecta al servidor C&C HTTP del dominio especificado en el archivo de configuración, utilizando un mensaje HTTP POST en el que viaja la información personal robada, probablemente información sobre las cuentas bancarias del usuario infectado.

Estudios de características de las botnets *en general*

En lugar de centrarse en una *botnet* concreta, existen trabajos que pretenden aportar luz en la comprensión del comportamiento general de un conjunto de *botnets* o de *botnets* de un cierto tipo, en relación a algunas características concretas: tamaño, distribución geográfica y temporal de sus *bots*, etc. Este es el caso de [134], trabajo en el que los autores indican algunos retos que surgen al estudiar el comportamiento general de las *botnets* con el objetivo de medir el tamaño, la topología y las dinámicas de las *botnets* distribuidas. Entre estos retos se resalta la existencia de tráfico generado por aplicaciones que no están relacionadas con la *botnet*, o la presencia de otros nodos activos en la red que puedan influir en el funcionamiento general de la misma.

En [135] los autores presentan PPM (*Passive P2P Monitor*), un sistema de monitorización capaz de enumerar los equipos infectados sin importar si están tras un NAT (*Network Address Translation*) o no. Rajab et al. [7] construyeron una infraestructura de medida distribuida que utilizaron durante un periodo superior a 3 meses para seguir 192 *botnets* IRC. Estos autores revelaron características estructurales y de comportamiento de estas *botnets*. Posteriormente, los mismos autores mostraron en [136] cómo ciertos aspectos, que incluyen clonación, migración temporal y estructuras ocultas, pueden incrementar significativamente la dificultad de determinar el tamaño de una *botnet* con precisión.

Diseño de nuevas *botnets*

Finalmente, algunos artículos se han centrado en definir alternativas posibles para el diseño de nuevas *botnets*. Estos estudios permiten que la comunidad investigadora pueda desarrollar mecanismos de defensa incluso antes de que las amenazas aparezcan. Por ejemplo, los autores de [86] diseñaron una *botnet* con arquitectura híbrida compuesta por algunos supernodos que actúan como servidores C&C para otras *botnets* distribuidas. Jian et al. [137] presentaron un método para seleccionar una lista de vecinos utilizando *Strongly Connected Graph* y así analizar un mecanismo de selección de supernodo basado en AHP (*Analytic Hierarchy Process*). El diseño de una *botnet* P2P con criptografía y un sistema de créditos se presentó en [138].

Starnberger et al. [139] diseñaron Overbot, un nuevo protocolo de comunicación de *botnets* P2P basado en Kademlia, el algoritmo DHT (*Distributed Hash Table*), para realizar las comunicaciones C&C. Los mensajes entre los *bots* se codifican utilizando las funciones `FIND_NODE` y `FIND_VALUE` que permiten a un nodo buscar un valor específico en la DHT. Overbot puede utilizar los *hash* de 160-bits que son parte de las solicitudes de búsqueda para transmitir comandos a sus *bots*. Esta *botnet* es de tipo parásito ya que su funcionamiento se basa en las redes P2P distribuidas BitTorrent o eDonkey. Nappa et al. [140] presentan el diseño de una *botnet* basada en Skype. Finalmente, Wang et al. [141] presentan un mecanismo que puede ser utilizado tanto por *botnets* centralizadas como distribuidas para detectar *honeypots*. Dado que los *honeypots* no están autorizados a lanzar ataques, esta detección se basa en la identificación de las máquinas que no ejecutan ataques aún cuando así se les ordena.

5.4.2. Reclutamiento

Se han realizado muchas propuestas por parte de la comunidad investigadora para prevenir la infección de equipos en Internet, con una atención especial a la propagación de virus y gusanos. Aplicado al problema de las *botnets*, y de acuerdo

a la taxonomía aquí propuesta, todas estas técnicas se localizan en la etapa de reclutamiento. Las contribuciones más relevantes que se presentan a continuación se dividen en dos grupos: (*i*) estudios del proceso de reclutamiento y (*ii*) técnicas de defensa contra el reclutamiento. Ambos se discuten a continuación.

Estudios del proceso de reclutamiento

El proceso de reclutamiento propio de las *botnets* ha sido estudiado por muchos proyectos de investigación a fin de extraer características útiles que faciliten el desarrollo de técnicas de detección. Estos estudios ponen de manifiesto que la mayor contribución de intentos maliciosos de conexión se atribuye directamente a la actividad de propagación de las *botnets*. De hecho, según Rajab et al. en [7] alrededor del 27 % de todos los intentos maliciosos de conexión son debidos a las acciones de reclutamiento de *botnets*. Li et al. [142] analizaron gran cantidad de tráfico, determinando la relevancia de los procesos de reclutamiento de las *botnets*. Este análisis, realizado utilizando datos de una *honeynet*, es capaz de extraer información de forma local acerca de los procesos de escaneo de una *botnet* y, posteriormente, extrapolar características a un plano más global. Es importante resaltar el trabajo de Caballero et al. [143] en el que realizan un estudio de los servicios PPI (*Pay-Per-Install*) infiltrándose en varios de ellos. En este estudio encontraron que 12 de las 20 familias más importantes de *malware* de 2010 hacían uso de servicios PPI para reclutar a sus miembros.

En otro tipo de estudios se proponen modelos de propagación para la etapa de reclutamiento. Por ejemplo, Dagon et al. [144] extraen un modelo de propagación diurna para examinar cómo afectan el tiempo y la localización geográfica a las dinámicas de propagación de las *botnets*. Otro ejemplo es [145], en el que los autores construyen un modelo de propagación de la *botnet* Conficker. Este modelo toma en consideración la localización y la conectividad entre los *bots*. Otra aproximación en el estudio del reclutamiento es estudiar y comparar los métodos de propagación de *botnets* específicas. Este es el caso de [146], que estudia los métodos de infección y propagación utilizados por las *botnets* SDBot, Rbot, Agobot, Phatbot, Spybot y Mytob.

En la actualidad es tremendamente común la infección por descarga dirigida (*drive-by-download*), en la que un usuario al visitar una página web o un correo electrónico descarga e instala un programa sin ser notificado. Como Provos et al. muestran en [93], alrededor del 1,3 % de todas las solicitudes de búsqueda de Google dan como resultado al menos un enlace a una página que realizar un ataque *drive-by-download*. En [147] los autores estudian los 4 mecanismos básicos utilizados para inyectar tráfico malicioso en sitios web populares: seguridad del servidor web, contribuciones de contenido por parte del usuario, publicidad y *widgets* de terceras partes. Más tarde, en [148] se propone un ciclo de vida de *malware* basado en web.

Para capturar el tráfico de red de las máquinas infectadas, los autores implementan un software capaz de responder automáticamente a los protocolo más utilizados: SMTP, FTP e IRC. En [149] los autores estudian una gran cantidad de datos de infección de Conficker, MegaD y Srizbi con el objetivo de encontrar similitudes y diferencias entre los miembros de las *botnets* con capacidades de autopropagación y los miembros de las *botnets* que necesitan métodos externos para propagarse, *p.ej.*, *drive-by-download* o PPI.

Técnicas de defensa contra el reclutamiento

En el campo de las defensas contra la etapa de reclutamiento, la mayoría de las contribuciones relacionadas con *botnets* se enfocan en la detección de este tipo de eventos. Jordan et al. [150] proponen recopilar y analizar los patrones de crecimiento de una *botnet* a través de una monitorización a nivel de red. Así, se puede interceptar la transmisión del *malware* en su proceso de infección, capturándolo en una etapa muy temprana. Gu et al. [151] presentan BotHunter, una estrategia de monitorización de red que se basa en reconocer las operaciones propias de la infección de un *bot* y, tras éstas, se inicia un proceso de monitorización de la actividad de red del equipo infectado. En cuanto a los ataques *drive-by downloads*, Egele et al. proponen en [152] un mecanismo de detección integrado en el navegador del usuario y basado en la emulación de instrucciones x86. Este sistema es capaz de identificar cadenas JavaScript con *shellcode*.

5.4.3. Interacción

Dado que la existencia de comunicaciones C&C es una propiedad inherente a las *botnets*, multitud de investigadores han empleado su tiempo y esfuerzo en descubrir medidas de defensa enfocadas en la etapa de interacción. De hecho, la mayoría de los artículos que implementan mecanismos de defensa frente a *botnets* se centran en las interacciones de éstas. Para revisar la ingente cantidad de trabajos de investigación en este campo con un cierto grado de orden, éstos se han clasificado en los siguientes 4 grupos: (*i*) detección de comunicaciones C&C basadas en el tráfico de red, (*ii*) detección basada en análisis de los protocolos de comunicación, (*iii*) detección de comunicaciones con servicios externos y (*iv*) mecanismos de respuesta contra *botnets*.

Detección de comunicaciones C&C basadas en el tráfico de red

Las contribuciones que se encuentran en este grupo están relacionadas con la detección de mensajes C&C sin tomar en consideración el protocolo de comunicación utilizado.

Algunas aproximaciones [153] [154] [155] son capaces de detectar el tráfico relacionado con mensajes C&C sin requerir conocimiento a priori de las *botnets*, o

incluso sin necesidad de especificar el protocolo de comunicaciones o la estructura de red. En [153], los autores utilizan periodogramas para estudiar el comportamiento periódico de estas comunicaciones y aplican el test de Walker [156] para averiguar cuando el tráfico analizado tiene una componente periódica significativa o no. La hipótesis de los autores asume que si el tráfico presenta una componente periódica significativa es tráfico de comunicaciones C&C de una *botnet*. Pham et al. [154] proponen un método para identificar y agrupar trazas de red recopiladas en *honeypots* de baja interacción por equipos que pertenecen a una *botnet*.

El entorno de detección BotMiner [155] agrupa tráfico normal y tráfico malicioso similares. Tras esto, realiza correlación cruzada de los grupos resultantes, a fin de identificar los equipos con patrones de comunicación y patrones de actividad maliciosa similares. En [157] los autores presentan un mecanismo automático para generar modelos de detección de *bots*, basados en la observación del comportamiento de *bots* reales instanciados en un entorno controlado. Estos modelos se basan en las firmas del tráfico de red asociadas a la respuesta generada por un *bot* al recibir el comando enviado por el *botmaster*.

En [98] los autores proponen una aproximación llamada BotSniffer que utiliza detección de anomalías a nivel de red. En este caso, la estructura de red se asume centralizada. BotSniffer identifica canales C&C de *botnets* centralizadas en una red de área local sin ningún conocimiento a priori de firmas o direcciones de servidores C&C. Esta aproximación de detección puede identificar los servidores C&C y los equipos infectados en la red y se basa en la correlación espacio-temporal y la similitud de *bots* pertenecientes a las mismas *botnets*. Xiaocong et al. [158] proponen transformar los flujos de tráfico de red en flujos de características multidimensionales en una ventana temporal, después de lo cual se agrupan flujos de características con elevada similitud. Los equipos clasificados en este grupo se etiquetan como *bots* en potencia. Esta aproximación es suficientemente eficiente como para ser utilizada online. Otros autores [159] proponen esquemas para detectar canales C&C de *botnets* P2P modelando perfiles de comportamiento de nodos en base a correlaciones espaciales y temporales. Strayer et al. [160] utilizan el ancho de banda, el tiempo de paquetes y la duración de las ráfagas para detectar actividad de mensajes C&C de una *botnet*.

Algunos trabajos combinan características de una red con otras de diferentes orígenes. Por ejemplo, los autores de [161] presentan un entorno que combina la información de nivel de red y de nivel de *host* para determinar cuándo el equipo analizado es parte de una *botnet* y cuándo no. EFFORT (*Efficient and Effective Bot Malware Detection*) [162] sigue una aproximación similar en la que se correla información de nivel de *host* y de red de 5 módulos diferentes. Otros autores [163] sugieren la combinación de características de red extraídas de diferentes localizaciones geográficas.

Detección basada en análisis de los protocolos de comunicación

En las contribuciones de esta categoría se analizan las comunicaciones C&C tomando en consideración las particularidades de los protocolos utilizados en ellas. La mayoría de los artículos en la bibliografía se centran en tres protocolos de comunicación: IRC, HTTP y P2P.

Existe una gran cantidad de trabajos de investigación que pretenden detectar *botnets* cuyas comunicaciones C&C viajan a través del protocolo IRC. Esto se debe probablemente a que, como se comentó anteriormente, IRC fue el primer protocolo utilizado en las comunicaciones de *botnets* y, aún hoy, algunas *botnets* lo continúan utilizando. Mazzariello [164] propone un entorno para la detección de la actividad de *botnets* por medio de un modelado del comportamiento de los usuarios de IRC. Este entorno es capaz de detectar y decodificar actividad de IRC en trazas de tráfico real y construir un clasificador a fin de diferenciar instancias de actividad de IRC normal de aquéllas relacionadas con la actividad de *botnets* IRC. En [165] los autores proponen un algoritmo llamado "distancia de canal", basado en la similitud de alias virtuales (*nicknames*) en el mismo canal, para detectar *botnets* IRC. De forma similar, Goebel et al. [166] implementan Rishi, un sistema cuyo objetivo es detectar *bots* IRC utilizando técnicas basadas en monitorización pasiva de tráfico de red en búsqueda de *nicknames*, servidores IRC o puertos de los servidores poco frecuentes o sospechosos. Para complementar estas estrategias de detección pasiva de *botnets*, más en concreto para *botnets* de un tamaño reducido, con los mensajes C&C ofuscados y con interacciones poco frecuentes, los autores de [167] presentan el entorno BotProbe. Este entorno utiliza técnicas activas para separar interacciones C&C de conversaciones humanas. Binkley et al. [168] contribuyen con un algoritmo de detección basado en anomalías. Este algoritmo se basa en dos tuplas, una para determinar mallas IRC en base a nombres de canales IRC y otra que recopila estadísticas del número de paquetes `SYN`, `FIN` y `RESET` observados.

En lo relativo a detección de *botnets* HTTP, Chen et al. [169] desarrollan un mecanismo basado en flujos de tráfico web anómalos en una red administrativa. Este trabajo se basa en la idea de que los *bots* web exhiben patrones repetitivos de conexiones que pueden ser utilizados para identificar flujos de tráfico web poco usuales dentro de una red. Lee et al. [170] muestran la relación entre clientes y servidores HTTP y también proponen un método para identificar *botnets* HTTP detectando patrones con cierta periodicidad.

Finalmente, existen algunos artículos relacionados con la detección de *botnets* que se comunican a través de protocolos P2P. Por ejemplo, Kang et al. [171] proponen un método de detección que utiliza el algoritmo multi-chart CUSUM (*CUMulative SUM*). Este trabajo es posteriormente mejorado y ampliado en [172], en el que se describe un modelo de detección de tiempo real, KCFM (*Kalman filter and multi-chart CUSUM Fused Model*). Este algoritmo usa el filtro discreto de Kalman para localizar tráfico anómalo y el multi-chart CUSUM actúa como amplificador para realzar la anormalidad. El objetivo de Masud et al. en [173] es detectar *botnets* P2P utilizando

minería de datos de red. Una aproximación similar es realizada por Liao et al. [174]. En la misma línea, Nagaraje et al. proponen en [175] BotGrep, un algoritmo que aísla las comunicaciones P2P basado en la interconexión de las comunicaciones entre nodos (gráfico de comunicaciones).

Los autores de [176] basan su método de detección de *bots* P2P en la identificación de conexiones de red con miembros conocidos de una *botnet*. Ellos muestran que una vez conocido un subconjunto de *bots* de una *botnet* es posible extraer un elevado porcentaje de los miembros de la misma en función de las comunicaciones en común que presentan los *bots* conocidos. Zhang et al. en [177] presentan un sistema capaz de detectar *botnets* P2P con una actividad maliciosa a baja tasa, a estas *botnets* las llaman *botnets* P2P sigilosas (*stealthy P2P* botnets). Para esto, primero hallan los equipos que reflejan un comportamiento P2P. De éstos, extraen huellas estadísticas de las comunicaciones P2P que generan. Finalmente, en función del solapamiento entre las direcciones IP contactadas por los nodos y si las conexiones son persistentes o no, diferencian entre equipos normales que usan P2P y *bots* P2P.

Detección de comunicaciones con servicios externos

Las contribuciones de este grupo se relacionan con la detección de actividades generadas por una *botnet* mientras ésta interactúa con servicios externos no comprometidos. En esta línea, muchas de las aproximaciones se basan en la detección de comunicaciones de una *botnet* con servidores DNS. En [178] los autores evalúan dos aproximaciones para identificar servidores C&C basadas en tráfico anómalos DDNS (*Dynamic Domain Name System*). La primera, consiste en localizar nombres de dominio cuyas tasas de envío de consultas sean anormalmente elevadas o estén muy concentradas temporalmente. La segunda, busca la recurrencia en los mensajes de respuesta DDNS que indican que la solicitud está buscando un nombre de dominio que no existe (NXDOMAIN). Yadav et al. proponen en [179] un algoritmo para detectar cambios frecuentes de dominio (*domain* flux) en el tráfico DNS. Para esto, buscan patrones dentro de los nombres de dominio que revelen que han sido generados con un algoritmo. Utilizan la distribución de caracteres alfanuméricos y los bigramas de los dominios mapeados para el mismo grupo de direcciones IP.

En [180] se utiliza el trabajo previo en conjunción con una aproximación similar a [178]. Aquí, el método de detección se basa en la correlación entre mensajes de respuesta NXDOMAIN y la entropía de los dominios pertenecientes a las solicitudes DNS. Choi et al. [181] proponen un mecanismo de detección basado en la monitorización del tráfico de las consultas DNS para detectar aquéllas que son enviadas simultáneamente, asumidas como generadas por *bots* distribuidos. Ramachandran et al. [99] se basan en la idea de que los *botmasters* realizan búsquedas en DNSBL para determinar si sus *bots* de *spam* han sido bloqueados (introducidos en una *blacklist*) o no.

Mecanismos de respuesta frente a *botnets*

Claramente, los mecanismos de respuesta frente a *botnets* más comunes se basan en una detección inicial para, posteriormente, filtrar el tráfico que viene desde direcciones IP detectadas como *bots* (aproximación no escalable), o bien impedir el servicio de los servidores C&C dejando a los *bots* sin órdenes que ejecutar.

En consecuencia, el grueso de las contribuciones en defensa frente a *botnets* está enfocado principalmente a la detección. Sin embargo, existen algunos artículos que se centran específicamente en el desarrollo de mecanismos de respuesta. Por ejemplo, algunas propuestas para respuesta frente a *botnets* P2P se basan en el uso de un ataque *sybil* para neutralizar la *botnet*. Aquí, se crea un enorme número de entidades virtuales que se introducen en la red P2P de la *botnet* para ganar influencia en ella. Algunas de las contribuciones más relevantes en esta línea son [130] [182] y [183].

En primer lugar, Holz et al. [130] presentan un caso de estudio en el que muestran cómo utilizar el ataque *sybil* para infiltrarse en la *botnet* Storm. Uno de los principales objetivos de este estudio fue determinar el efecto de los ataque de polución (envenenamiento de índices), publicando valores erróneos de claves asociados con las búsquedas de Storm. Así, los autores evaluaron la efectividad del ataque de polución ejecutándolo en las claves utilizadas por la *botnet* Storm y, usando un programa capaz de explorar toda la red Overnet, buscaron esas claves. Su experimento demostró que llevando a cabo un ataque de polución exitoso contra las claves de Storm es posible inhabilitar las comunicaciones de la *botnet* y, por tanto, inutilizar la *botnet* completa.

Los trabajos de Davis et al. [182] y [183] son complementarios al realizado por Holz et al. [130]. Una clave de los ataques de envenenamiento de índices es que los nodos *sybil* deben continuar activamente participando en los protocolos P2P para permanecer como vecinos de los *bots*. Sin embargo, al realizar esta detección no pueden responder a las órdenes del *botmaster* o participar en actividades ilícitas. Los estudios de Davis et al. pretenden descubrir cómo de resistentes pueden ser las *botnets* frente a las contramedidas presentadas por Holz et al. utilizando esta idea como una contra-contramedida. Otro trabajo en esta línea es [184], en el que los autores simulan una *botnet* P2P basada en Kademlia y evalúan la efectividad de las técnicas de mitigación potenciales frente a ella: envenenamiento de contenido, basadas en el ataque *sybil* y basadas en el ataque eclipse [185].

5.4.4. Marketing

Los mecanismos de defensa agrupados en la etapa de marketing del ciclo de vida pueden ser altamente efectivos en la neutralización del funcionamiento de las *botnets*. Tomar medidas legales en contra del marketing de *botnets* es una aproximación

posible. Obviamente, si aquéllos que anuncian los servicios de *botnets* estuvieran sujetos a penas legales, esto podría disminuir los anuncios de *botnets* y, por ende, el uso de las mismas. Las medidas para evitar el desarrollo satisfactorio de la etapa de marketing deben enfocarse también en capturar e identificar actividades de los *botmasters*. Considérese, por ejemplo, el caso de Thomas James Frederick Smith en [101], uno de los creadores de la *botnet* IRC *NETTICK*. Publicó un mensaje en multitud de foros en el que ofrecía un ejecutable para controlar su *botnet* por 750 dólares. El 10 de junio de 2010 fue declarado culpable por conspirar para provocar daño de forma intencionada a equipos protegidos.

Como ejemplo de una contribución en la cual los investigadores aprovechan la etapa de marketing con fines de detección se puede mencionar el trabajo [99], en el que los autores detectan una *botnet* inspeccionando las búsquedas realizadas a DNSBL, páginas en las que los dominios que se encuentran en una lista negra (*blacklisted*) están públicamente disponibles. Estas búsquedas se realizan por parte de los *botmasters* para asegurarse la disponibilidad de sus *botnets*, probablemente antes de alquilar sus servicios.

La mayoría de las contribuciones de la comunidad investigadora se centran en la comprensión de esta nueva economía clandestina, ya que esto es un primer paso crucial para diseñar esquemas de defensa apropiados que permitan frustrar la etapa de marketing. En lo que sigue se presentan algunas contribuciones relevantes en la línea de conseguir un mejor entendimiento de este mercado agrupadas en dos: (*i*) análisis de los anuncios y (*ii*) análisis de monetización.

Análisis de los anuncios

La primera exploración de esta economía clandestina en actividades como fraude de tarjetas de crédito, robo de identidad y envío de *spam* fue llevada a cabo por Franklin et al. en [186]. En este trabajo los autores analizaron los mensajes intercambiados en *chats* públicos de Internet durante 7 meses y resaltaron la importancia del desplazamiento del "*hacking* por diversión" al "*hacking* por beneficio". Más tarde, estos mercados evolucionaron moviéndose hacia foros web y en los que también se comparte información acerca de actividades maliciosas. En [187] se presenta una revisión detallada de este mercado clandestino y un modelo para describirlo. Los autores espían un foro de este mercado negro para estimar el volumen de actividad criminal.

Algunas contribuciones analizan el comportamiento de los participantes de estos foros. Así, en [188] los autores resaltan la interacción, reglas y comportamiento social de los participantes. En esta misma línea, un sistema capaz de monitorizar automáticamente estos canales y sus participantes se presenta en [189]. Este sistema monitoriza tanto foros basados en IRC como en web. Motoyama et al. examinan

empíricamente en [190] las redes sociales formadas en estos foros y los mecanismos que se emplean para gestionar la confianza, caracterizando la formación de las redes sociales de 6 foros clandestinos: BlackHatWorld, Carders, HackSector, HackE1ite, Freehack y L33tCrew.

Otros autores proponen tomar como base las defensas empleadas contra otros mercados negros. Este es el caso de [191], trabajo en el que los autores proponen el uso de mecanismos que han sido eficientes en la investigación policial en la lucha contra el tráfico de drogas y se señalan algunas acciones que podrían aprenderse de este campo. Es el caso del uso de un sistema similar a STRIDE (*System to Retrieve Information from Drug Evidence*) [192], el cual monitoriza las transacciones económicas relacionadas con las drogas.

Análisis de monetización

Multitud de contribuciones se centran en la cuestión de la cuantificación de la repercusión económica (monetización) de un servicio en particular, como el de robo de información personal o el envío de *spam*. Holz et al. en [193] capturaron y analizaron un total de 33 GB de datos de *keylogger* instalados en más de 173.000 equipos comprometidos. Los autores se centraron en cuestiones acerca del tipo y la cantidad de información robada y concluyeron que este tipo de cibercrimen es un negocio que presenta un gran beneficio económico, permitiendo a los atacantes ganar cientos de miles de dólares al día. Por ejemplo, recuperaron más de 10.700 cuentas bancarias robadas, que potencialmente pueden ser vendidas en el mercado clandestino por varios millones de dólares.

En cuanto al mercado de *spam*, los autores de [194] se infiltraron en la infraestructura de una *botnet* existente. Fueron capaces de identificar el número de visitas conseguidas a los sitios de publicidad anunciados, el número de "ventas" y el número "infecciones" producidas por cerca de 500 millones de correos *spam*. Este trabajo permite diferenciar entre las pérdidas supuestas y las reales. En [195] se extiende el trabajo anterior y los autores analizan grandes campañas de correo *spam* examinando la totalidad de la cadena de mercado; sin embargo, este estudio ofrece menos precisión en cuanto a los costes específicos. En [196] los autores describen dos técnicas de inferencia para investigar la publicidad relacionada con el correo *spam*: (*i*) estimar el número de solicitudes recibidas y, por tanto, el beneficio, a través de la numeración de los pedidos de las tiendas online y (*ii*) caracterizar el comportamiento de las compras a través de terceras partes de hospedaje de información de imágenes. Utilizando esta información, son capaces de recabar información del volumen de las solicitudes, de la distribución de las ventas de productos, de la creación de clientes y de los beneficios totales de varias campañas publicitarias de *spam*.

5.4.5. Ejecución del ataque

Como se describió en la Sección 5.3.5, los ataques más comunes ejecutados por *botnets* son: DDoS, *spam*, *phishing* y fraude del *click*. Se han publicado muchas contribuciones en la línea de cómo luchar contra estos ataques, en lo relativo a prevención, detección y respuesta, pero aún existen muchas cuestiones que continúan sin ser resueltas.

Aunque existen multitud de contribuciones que proponen contramedidas frente a estos tipos de ataques [197] [198], esta sección se centra en aquellos trabajos que están directamente relacionados con las contramedidas frente a los ataques mencionados pero en el campo de las *botnets*.

El entorno de detección Botminer [155] es capaz de identificar como *bots* a todos los equipos que comparten patrones similares de actividades maliciosas y de comunicación.

Los autores de [113] utilizan el tráfico generado por los ataques DDoS para detectar miembros de las *botnets*. Esta detección se basa en la constatación de que la actividad automática y coordinada de muchos equipos precisa de un mecanismo para controlarlos remotamente.

También en [199] se utilizan los patrones de ataque del fraude del *click* para detectar miembros de una *botnet*. Los autores presentan SBotMiner, un sistema para identificar automáticamente tráfico de *bots* sigilosos (que generan poco tráfico) que realizan búsquedas a una baja tasa. Esta aproximación es también capaz de detectar grupos de *bots* que realizan búsquedas coordinadas y distribuidas. Duan et al. [200] utilizan la información de la actividad del correo *spam* para detectar miembros de una *botnet* y desarrollan un sistema de detección de *bots* de *spam* llamado SPOT, el cual monitoriza los mensajes de salida de una red. El diseño del sistema SPOT se basa en una herramienta estadística, *Sequential Probability Ratio Test*, gracias a la que las tasas de falsos positivos y falsos negativos están acotadas. De la misma forma, los autores de [201] caracterizan *botnets* de *spam* utilizando el contenido de los correos y las propiedades de los servidores de tráfico *spam*, a través de un entorno para la generación de firmas llamado AutoRE. BotGraph [202] es otro sistema de detección de *botnets spam* capaz de detectar usuarios sigilosos en dos pasos. Primero, se detectan los inicios de sesión agresivos (un gran número de usuarios en poco tiempo), limitando así el número total de cuentas que posee el *spammer*. Segundo, una aproximación basada en grafos se utiliza para detectar las cuentas de *bots* sigilosas, a partir de la correlación entre las actividades de inicio de sesión.

Duwairi et al. [203] sugieren una detección de DDoS utilizando la información de los motores de búsqueda web (especialmente *Googletrade*). La idea principal es

que Googletrade puede ayudar a distinguir usuarios humanos de programas *bots*, mediante la redirección de los usuarios humanos que quieren acceder a una página que está siendo atacada hacia un grupo de equipos en los que se realice un proceso de autenticación por medio de test de Turing. En [204] Collins et al. predicen actividad en base a la actividad de red pasada. Para hacer esto, definen una medida de la calidad de la limpieza de una red. Esta medida es un indicador de la probabilidad de que una red contenga equipos comprometidos. En este estudio los autores muestran también que la actividad de *botnet* predice *spamming* y actividades de escaneo.

5.4.6. Mecanismos complementarios de ocultación

El software malicioso utiliza generalmente mecanismos complementarios de ocultación, ya que esto dificulta su detección; y las *botnets* no son una excepción. La comunidad investigadora ha publicado muchos trabajos dirigidos a detectar y contrarrestar estos mecanismos de ocultación.

Como se mencionó en la Sección 5.3.6, los mecanismos de ocultación más comunes utilizados por *botnets* son: utilización de conexiones multi-salto, cifrado, ofuscación de binario, polimorfismo, falseamiento de direcciones IP y de correo y redes *fast-flux*. Un tipo de mecanismos de ocultación de especial relevancia sigue la línea del polimorfismo y la ofuscación de binarios, y es denominado *packers/cryptors*. Un *packer* [205] es un programa utilizado para comprimir y encriptar ficheros ejecutables almacenados en disco, y restaurar la imagen del ejecutable original cuando éste es cargado en memoria. Los packers se utilizan para proteger las aplicaciones de su análisis mediante ingeniería inversa. Aunque esto puede utilizarse en aplicaciones con derechos de autor, para dificultar que los atacantes puedan, por ejemplo, instalarlas sin utilizar una clave de instalación, también pueden ser utilizadas por los autores de *malware* para ofuscar la estructura de sus programas, dificultando por tanto las tareas de detección. Un archivo empaquetado con diferentes packers resulta en binarios diferentes, lo que impide su detección por sistemas de detección basados en firmas y, además, complica enormemente las tareas de ingeniería inversa. Como muestra el estudio [206], más del 92% de los 735 *malware* analizados durante 2006 fueron empaquetados con un packer. En el mismo trabajo, los autores aseguran que el uso de packers en un *malware* puede reducir considerablemente la posibilidad de su detección (alrededor de un 40% en algunos casos). De hecho, los *malware* empaquetados por algunos packers no son detectados por algunos sistemas antivirus, *p.ej.*, UPX, Armadillo, Themida, VM-project, Epack, Cexe, etc. Los autores de [207] también declaran que la efectividad a largo plazo de los antivirus basados en *hosts* es cuestionable, debido, entre otras causas, a las técnicas actuales de polimorfismo.

En lo que resta de sección nos centramos en las redes *fast-flux* dado que la implementación de esta técnica precisa necesariamente el uso de una *botnet*, mientras que

las demás técnicas son comunes a otros muchos tipos de *malware*. El uso de *botnets* en las redes *fast-flux* es clave porque estas redes requieren de un elevado número de equipos comprometidos para su implementación, estos equipos comprometidos son los miembros de la *botnet*. Tomando lo anterior como cierto, la detección de participantes en una red *fast-flux* resulta en el mismo problema que la detección de miembros de una *botnet*.

El primer caso de estudio de redes *fast-flux* se presentó en [208], en el que los autores describieron multitud de redes de este tipo. Este artículo también proporcionó una visión general sobre qué eran realmente las redes *fast-flux*, cómo funcionaban y cómo las comunidades clandestinas hacían uso de ellas. Posteriormente, apareció FluXOR [209], que fue el primer sistema de detección de redes *fast-flux*. Las estrategias de detección de FluXOR se basan en el análisis de un conjunto de características observables desde el punto de vista de un usuario ordinario. Los autores proponen tres tipos de características para distinguir entre nombres de equipos legítimos y maliciosos: (*i*) nombre de dominio, (*ii*) disponibilidad de la red y (*iii*) heterogeneidad de los agentes. Holz et al. en [210] definen una métrica para diferenciar redes *fast-flux* de redes de distribución de contenido legítimo. Esta métrica se basa en dos restricciones inherentes a las redes *fast-flux*: (*i*) el rango de direcciones IP de una red *fast-flux* es bastante diverso y (*ii*) no hay garantías de que los agentes *flux* estén activos, ya que los equipos comprometidos pueden ser apagados en cualquier momento.

Nazario et al. [211] utilizan técnicas de minería de datos de tráfico de red para descubrir nuevos dominios *fast-flux* para posteriormente, seguir aquellas *botnets* activas durante meses para estudiarlas. Alper et al. [212] presentan un análisis del comportamiento de las redes *fast-flux* utilizando una base de datos generada por ellos mismos durante 9 meses. Sus resultados muestran que estas redes comparten características en el ciclo de vida y forman grupos en base al tamaño, crecimiento y tipo de comportamiento malicioso.

Caglayan et al. presentan en [213] el primer estudio empírico para detectar y clasificar redes *fast-flux* en tiempo real. En [214] los autores proponen una aproximación pasiva para detectar y seguir redes *fast-flux*. Su sistema de detección se basa en el análisis pasivo de las trazas de tráfico de los DNS recursivos (RDNS; *Recursive Domain Name System*) recopiladas desde múltiples redes de gran tamaño. Este método es capaz de detectar redes *fast-flux* maliciosas desde los datos que éstas generan y no desde correos *spam* o dominios bloqueados (*blacklisted*). Leyla et al. presentan en [215] EXPOSURE, un sistema que emplea técnicas pasivas de análisis de tráfico DNS a gran escala para detectar dominios involucrados en actividades maliciosas. Los autores caracterizan diferentes propiedades de los nombres DNS y de las formas en las que son consultados, extrayendo del tráfico DNS hasta 15 características agrupadas en los siguientes conjuntos: características basadas en tiempo, características basadas en las respuestas DNS, características basadas en el valor del TTL y características basadas en el nombre de dominio. En la misma línea que EXPOSURE, Antonakakis

et al. presentan en [216] Kopis, un sistema capaz de monitorizar flujos de consultas DNS y respuestas desde los DNS superiores en la jerarquía para detectar nombres de dominio maliciosos. La gran ventaja de Kopis es que puede detectar dominios maliciosos incluso cuando no hay información de reputación de IP.

Estas contribuciones muestran que los atacantes están adoptando las técnicas *fast-flux* y que existe aún mucho trabajo que hacer en este campo. Sugerimos que las redes *fast-flux* deben ser estudiadas en profundidad, pues, aunque no son un mecanismo complementario y frustrar su uso no neutralizaría la *botnet*, su detección puede ser una línea prometedora en la detección de *botnets* ya que pueden ayudar a detectar *bots* de ésta.

5.5. Líneas de investigación futura

A pesar del considerable número de propuestas realizadas y del trabajo llevado a cabo en el campo de la lucha contra *botnets*, nuestro estudio revela que aún existen ciertos aspectos que deben ser investigados en mayor profundidad. A continuación, se presentan los principales retos actuales a los que se enfrenta la comunidad investigadora en este campo.

5.5.1. Diseño de defensas frente a las nuevas amenazas

La revisión de las contribuciones en *botnets* que se ha presentado muestra que la detección de *botnets* es un tema ampliamente estudiado. Sin embargo, la detección y prevención de la aparición de nuevas *botnets* está aún por explorar.

Una nueva tendencia en el diseño de *botnets* es el uso de redes P2P legales para ocultar sus comunicaciones C&C. Estas nuevas *botnets* se denominan *botnets* parásitas [217] y presentan una nueva amenaza que puede ser especialmente dañina debido a su gran capacidad de ocultación. Storm [81] es un ejemplo de este tipo de *botnets*, la cual utiliza la red Overnet (red P2P distribuida basada en Kademlia) para encontrar sus *bots* y comunicarse con ellos.

Un trabajo de investigación en este tipo de *botnets* es [139], donde se presenta Overbot, un nuevo protocolo P2P basado en los existentes KAD y Kashmir (ambos basados en Kademlia). Otro ejemplo de *botnet* parásita es Alureon/TDL-4 [87], una de las *botnets* más activas en 2010, la cual se comunica a través de KAD, la red P2P distribuida del conocido cliente eMule.

En vista a la creciente importancia de estas nuevas *botnets*, es preciso desarrollar nuevos sistemas de detección y defensa frente a esta nueva amenaza. Este es el

objetivo fundamental del siguiente capítulo de esta tesis. En éste se diseña un sistema de detección de *botnet* parásitas P2P en base al comportamiento de compartición de los recursos que los *bots* utilizan.

Adicionalmente, se sugiere que la existencia de *botnet* parásitas ha de ser tenida muy en cuenta en el diseño de nuevos protocolos y sistemas. En la actualidad, los desarrolladores en este campo ya tienen presente la existencia de los ataques DoS y el problema de la infección de equipos en sus nuevos diseños. Sin embargo, es importante también promover mecanismos que prevengan el uso de canales C&C ocultos por parte de los desarrolladores de *botnets*.

5.5.2. Detección multi-etapa de *botnets*

A pesar del gran esfuerzo realizado en la línea de detección de *botnets*, este problema no está aún resuelto. El análisis del ciclo de vida de las *botnets* presentado muestra que cualquier mecanismo efectivo en la frustración de una de las etapas de éste constituye una defensa válida frente a las *botnets*. Sin embargo, en una línea similar a los paradigmas multi-capa propuestos recientemente para otras tecnologías, una buena aproximación puede basarse en esquemas multi-etapa para detectar *botnets*. Como ejemplo de esto, un mecanismo de detección de infección de *bots* (etapa de reclutamiento) podría ser combinado con una aproximación para detectar acciones publicitarias por parte de los desarrolladores de *botnets* (etapa de marketing).

BotMiner [155] es la primera aproximación que basa su detección en la asunción de que los *bots* se comunican con algún servidor o nodo C&C mientras realizan una actividad maliciosa. Los autores del trabajo utilizan en su detección la información de la etapa de interacción con la de ejecución del ataque. Sin embargo, como Zang et al. declaran en [177], las *botnets* modernas tienden a ser sigilosas en sus actividades maliciosas, lo que provoca que las aproximaciones de detección actuales sean ineficaces, incluyendo la anterior [155]. Por tanto, se precisan nuevas aproximaciones multi-etapa que se basen no sólo en las etapas de interacción y de ejecución del ataque, sino también en otras etapas del ciclo de vida.

5.5.3. Defensas frente a *botnets* en la etapa de marketing

Como se mencionó en la Sección 5.4.4, la mayoría de las contribuciones que se encuadran en la etapa de marketing se centran en el análisis de la monetización y la publicidad. Estos trabajos tienen como objetivo (entre otros) poner de manifiesto la importancia del problema de las *botnets*.

En este sentido, nosotros apuntamos, en base al estudio de la etapa de marketing, que los nuevos esquemas de detección frente a *botnets* han de intentar impedir la

etapa de marketing. Esto se debe a que en la actualidad las *botnets* han pasado del "*hacking* por diversión" al "*hacking* por beneficio" [186] y, por consiguiente, la etapa de marketing es un punto visible desde el que atacar a las *botnets* actuales.

Así, se deben desarrollar e implementar nuevas técnicas para detectar actividad maliciosa en un entorno publicitario: foros clandestinos, redes sociales, etc. En una fase posterior, las alertas derivadas de estas técnicas deben ser correladas con los vectores de ataque como nuevas campañas de *spam*, ataques DDoS o de fraude del *click*.

5.5.4. Mecanismos de respuesta frente a *botnets*

El número de trabajos enfocados en el desarrollo de mecanismos de respuesta contra *botnets* es también muy escaso. Esto puede deberse principalmente a la baja eficiencia de los mecanismos de detección actuales, al ser la detección una condición previa para la ejecución de una respuesta. Aún así, se han sugerido algunas aproximaciones en esta línea. Por ejemplo, en [218] los autores proponen unirse a la *botnet* y neutralizarla desde dentro, como sucede en el caso de una enfermedad autoinmune. Pese a este estudio y otros similares, la cuestión de la respuesta frente a *botnets* no ha sido explorada en profundidad y se precisa una investigación mucho mayor en ella.

Se podrían también desarrollar nuevas técnicas de respuesta basadas en técnicas de predicción existentes, de modo que, las *botnets* podrían ser detenidas desde el mismo momento de su aparición. Algunos ejemplos de estas técnicas de predicción son [144] [204] [149], si bien queda un largo camino que recorrer en esta dirección.

5.5.5. Evolución de la estimación del tamaño a la estimación del impacto

La mayoría de los estudios de medida de *botnets* son relativos al número de *bots* comprometidos por una *botnet* en particular, es decir, se busca una estimación de su tamaño. Como se sugiere en [219], el foco de la investigación debe evolucionar de la medida del tamaño de una *botnet* al análisis del impacto de *botnets* específicas de especial relevancia para la sociedad.

Es esencial analizar en profundidad las funcionalidades de las *botnets* y desarrollar nuevas aproximaciones para la identificación rápida de éstas. Algunas de las características de las *botnets* que han de ser constantemente analizadas son:

- Infraestructuras y protocolos C&C.

- Debilidades y vectores de ataque contra el *malware* e infraestructuras C&C que puedan utilizarse para contrarrestar sus ataques.
- Complejidad de los nuevos diseños de criptografía, técnicas de ocultación y ofuscación de código en el *malware*.

5.5.6. Revisión de métodos de infección basados en tecnologías emergentes

Un vector común para la etapa de reclutamiento se basa en explotar vulnerabilidades de software. Estas vulnerabilidades son especialmente reseñables en nuevos modelos de software o tecnologías. De acuerdo al informe técnico publicado por Juniper en febrero de 2012 [220], en 2011 el número de muestras de *malware* en el sistema operativo Android creció en un 3.000 %. Existen así nuevas *botnets* reportadas en Android, tales como Geinimi y Droid Kungfu, entre otras.

Debido a la gran cantidad de *malware* basado en las nuevas tecnologías, la comunidad investigadora debe considerar especialmente este tema con el objetivo de paliar, en la medida de lo posible, los efectos de las nuevas técnicas de infección.

5.6. Conclusiones

Debido a los efectos dañinos de las *botnets* y al considerable interés entre la comunidad investigadora en este campo, en este capítulo se propone una taxonomía para la investigación en *botnets* con un doble objetivo. Por una parte, describir el problema de las *botnets* desde una perspectiva global. Por otro lado, proporcionar una herramienta útil para organizar y clarificar la vasta cantidad y variedad de trabajos en este campo.

La taxonomía propuesta se basa en el propio ciclo de vida de las *botnets*, también introducido en este capítulo. Éste se ha diseñado con una idea central en mente: cada etapa del ciclo de vida debe ser satisfactoriamente completada para que la *botnet* consiga su objetivo, la ejecución del ataque para el que se implementó. Por tanto, interrumpiendo la ejecución de al menos una de las etapas del ciclo de vida se frustrará la actividad de la *botnet* completa o, cuando menos, se reducirán notablemente sus efectos.

Adicionalmente, en este capítulo se han revisado los trabajos de investigación más importantes en el campo de las *botnets*, los cuales han sido organizados en base al ciclo de vida anterior. De esta forma, también se ha mostrado que, como se aseguraba, las defensas propuestas por la comunidad investigadora se centran en una o más

etapas del ciclo de vida. Esta revisión se presenta aquí como una visión general de las contribuciones más importantes del campo.

Finalmente, el análisis de las contribuciones de investigación ha resultado en una imagen general que revela las deficiencias que aún precisan de una atención por parte de la comunidad en el futuro próximo. También se expone nuestro punto de vista particular sobre los retos de investigación, sugiriendo algunas líneas principales para enfrentarlos.

Capítulo 6

Detección de *botnets* parásitas en redes P2P basada en el comportamiento de la compartición de recursos

Como ya se comentó en el capítulo anterior, la detección de comportamientos asociados a actividad de *botnet* es un tema recurrente en la literatura de investigación actual. Así, el problema asociado a la detección de *botnets* P2P ha sido declarado como uno de los más difíciles en este campo. Además, como también se mencionó con anterioridad, dentro de las *botnets* existe una nueva amenaza cuya detección resulta aún más compleja: las *botnets parásitas*. Este tipo de *botnets* se diferencian de las *botnets* P2P tradicionales en que utilizan en su fase de interacción redes legítimas para comunicarse, lo cual posibilita la ocultación del tráfico malicioso y dificulta la detección.

En este capítulo se afronta la línea de detección de nuevas amenazas propuesta en el capítulo anterior. Concretamente se pretende diseñar un esquema de detección de *botnets* P2P parásitas basado en el modelado de la evolución del número de nodos que comparten un recurso en una red P2P durante el tiempo. Este sistema permite detectar comportamientos anormales asociados a recursos propios de estas *botnets*. Además, también se realiza una experimentación con tráfico real de la red Mainline, de la que se extraen resultados de detección prometedores, obteniendo patrones de comportamiento anómalo que pueden ser asociados a recursos propios de *botnets* P2P parásitas.

De forma resumida, nuestra propuesta de detección no radica en la determinación de comunicaciones específicas de tipo *botnet*, sino en comportamientos de compartición de recursos de esta naturaleza. Con este fin, la estructura de lo que resta de capítulo es la siguiente. En la Sección 6.1 se motiva la necesidad de una nueva aproximación para la detección de *botnets* parásitas. Los conceptos básicos de BitTorrent, Kademlia y Mainline, necesarios para la comprensión del presente trabajo, se exponen en la Sección 6.2. Un modelo de la evolución "normal" de la compartición de recursos en redes P2P se expone en la Sección 6.3. Además, en ésta se discute también un modelo teórico para el comportamiento esperado de los recursos compartidos por *botnets* P2P parásitas. En la Sección 6.4 se detalla el módulo para la monitorización de la evolución temporal de los recursos compartidos en la red Mainline. Tras esto, la arquitectura general del sistema de detección aquí propuesto es detallada en la Sección 6.5. En la Sección 6.6 se describe el entorno experimental utilizado, se analizan los recursos monitorizados y se realiza una evaluación del esquema de detección con los recursos monitorizados. Finalmente, las principales conclusiones del capítulo se resumen en la Sección 6.7.

6.1. Motivación

Como se comentó en el capítulo anterior, las *botnets* constituyen una seria amenaza contra sistemas, servicios y usuarios, ya que son uno de los principales vectores de infección de *malware* y permiten ejecutar una gran variedad de actividades perjudiciales (*p.ej.*, DDoS o *spam*) [221] [222].

De entre todos los tipos de *botnets*, las parásitas presentan una relevancia especial debido a que sus comunicaciones utilizan la infraestructura de redes legítimas. Este es el caso particular de las *botnets* P2P parásitas, en las que los mensajes enviados en la etapa de interacción de la *botnet* quedan ocultos dentro de una gran cantidad de tráfico legítimo de la red P2P que la transporta [223] [224].

La mayoría de las propuestas de detección se han centrado en introducir algún tipo de esquema de detección para extraer la actividad C&C de la *botnet* [225]. Sin embargo, una de las limitaciones principales de esta aproximación es que representar estas comunicaciones implica una especificación elevada, ya que el tráfico de red C&C varía mucho de unas *botnets* a otras. Como consecuencia, los mecanismos de detección desarrollados suelen caracterizar la actividad de una *botnet* concreta y no tienen capacidad de generalización. En relación a ello, estas propuestas requieren una gran cantidad de tráfico de red de *botnets* del que derivar modelos de comportamiento adecuados capaces de describir el comportamiento de la red. La obtención de este tráfico de red es una tarea de investigación en sí misma e intentar reproducir este tráfico en un ambiente controlado siempre entraña riesgos.

En el caso de las *botnets* P2P las comunicaciones C&C se basan en el intercambio de recursos compartidos por diferentes nodos de la red. Por ejemplo, el *botmaster* de una *botnet* P2P parásita puede crear un archivo cifrado que contiene las órdenes para los *bots* y compartirlo con ellos. Cada *bot* que descarga el archivo lo comparte a su vez con el resto de *bots* de la *botnet*. Nótese que las comunicaciones C&C aquí no difieren de la descarga de un recurso normal de la red P2P legítima y, por este mismo motivo, no es trivial construir un sistema de detección basado en el análisis del tráfico de red capaz de diferenciar la descarga de un recurso legítimo de la descarga de un recurso que contiene comandos del *botmaster*.

En este marco, este capítulo introduce una nueva aproximación para detectar *botnets* P2P parásitas basada en el comportamiento de la compartición de recursos en lugar de en la monitorización del tráfico de red. Específicamente, este sistema de detección se centra en la evolución temporal del número de nodos que comparten un recurso dado de la red. De esta forma se realiza, en primer lugar, un estudio del comportamiento normal de la evolución temporal de la compartición de recursos en redes P2P. En base a éste y a una serie de limitaciones a las que los *botmasters* se enfrentan al diseñar una *botnet*, también se presenta un modelo teórico del comportamiento esperado de la compartición de recursos por parte de una *botnet* P2P parásita. En este punto, estamos en disposición de plantear una propuesta de detección de posibles recursos *botnet* compartidos a través de una red P2P. Partiendo de la detección de estos recursos según [226], sería sencillo extraer una lista de los *bots* que están compartiéndolos, de modo que la operación normal de la *botnet* sería comprometida.

Esta es la primera vez, hasta la fecha, que se modela la compartición de recursos en lugar del tráfico de red para detectar actividades relacionadas con *botnets*. Adicionalmente a esto, las tres contribuciones principales del presente capítulo se indican a continuación:

- Una metodología para construir un modelo de compartición de recurso en función del tiempo para representar el comportamiento normal de una red P2P.
- Un modelo teórico de la compartición de recursos de *botnet* P2P parásitas y, basado en éste, un sencillo generador de tráfico de *botnets* de este tipo.
- Un sistema de detección para determinar la ocurrencia de eventos de *botnet* P2P parásitas.

Como caso de estudio, y principalmente motivado por su amplio uso, se presenta una experimentación exhaustiva para comprobar el buen funcionamiento del sistema de detección presentado en una red real de compartición de archivos, Mainline (red P2P estructurada de BitTorrent). Se obtienen resultados de detección prometedores

asociados con la aparición de patrones de evolución temporal de los recursos compartidos que pueden ser atribuidos a actividad de *botnets* P2P parásitas. Motivado por su empleo en experimentación, seguidamente se presentan brevemente los fundamentos de BitTorrent, particularizándose en Kademlia y Mainline.

6.2. Conceptos fundamentales de BitTorrent

BitTorrent es una red P2P de compartición de archivos. En ella se comparte principalmente software, películas, música, etc. Su objetivo fundamental es distribuir archivos de una forma eficiente a un elevado número de clientes, empleando los recursos que éstos ponen a disposición de la red (ancho de banda, almacenamiento, etc.) y motivando que aquellos que descarguen archivos los compartan con posterioridad. Los clientes en esta red se denominan nodos o *pares* (*peers*) y los archivos compartidos son llamados *recursos*.

El conjunto de todos los nodos que comparten un recurso común se denomina *enjambre* (*swarm*). Los nodos de la red se pueden clasificar en dos tipos: semillas (*seeders*) y sanguijuelas (*leechers*). Los *seeders* son los nodos que contienen una copia completa del archivo y los *leechers* son aquellos nodos que están descargando el archivo pero aún no tienen una copia completa.

Para poder descargar un recurso, un nodo de la red BitTorrent precisa de un archivo denominado *torrent*, que contiene información acerca del recurso a descargar. Esta información puede ser: el nombre del recurso, su tamaño, las partes en las que se divide el mismo, etc. Los archivos `.torrent` son descargados fuera de la red BitTorrent utilizando generalmente servidores web especializados. Algunos de los más comunes son The Pirate Bay [227] o Torrentz [228].

Así, la descarga de un recurso de la red BitTorrent consta de los siguientes pasos (véase la Figura 6.1):

1. *Elección del recurso*. El cliente busca el recurso que le interesa en los servidores web especializados, obteniendo en respuesta un archivo `.torrent` con toda la información que necesita para los siguientes pasos del proceso de descarga.

2. *Búsqueda de nodos*. En este paso el cliente busca en la red de BitTorrent una lista de nodos que compartan el recurso asociado al archivo `.torrent` descargado en el paso anterior.

3. *Intercambio de información*. Una vez que el cliente ha recibido las direcciones IP de los nodos que comparten el recurso, contacta con ellos para iniciar el proceso de descarga y compartición asociado. Este paso se realiza utilizando los mensajes propios del protocolo BitTorrent [229].

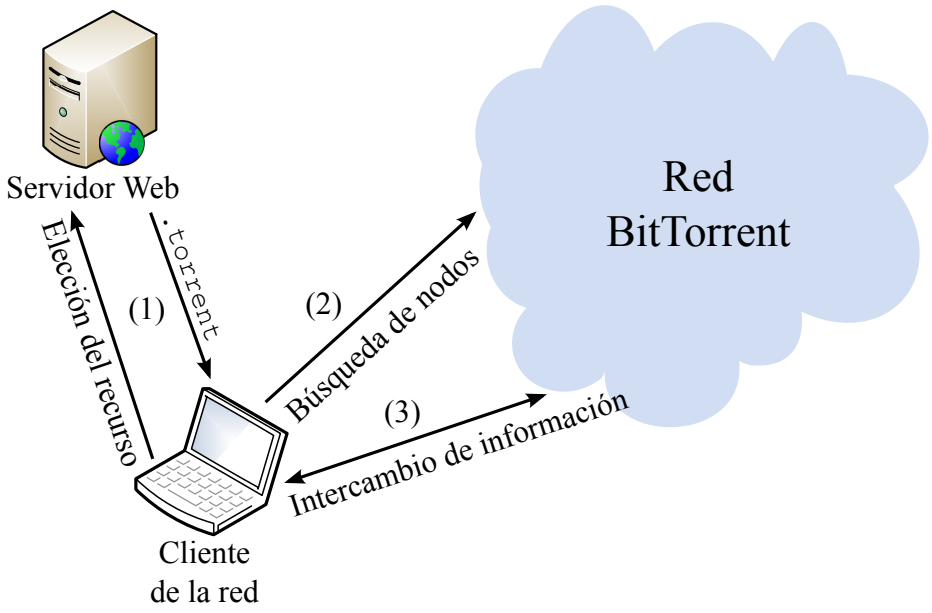


Figura 6.1: Esquema del proceso seguido por un nodo de la red BitTorrent para descargar un archivo.

El descubrimiento de nodos (paso 2) se realizaba originalmente solicitando a los *trackers* almacenados en el archivo `.torrent` la localización de los nodos que comparten el recurso buscado. El *tracker* es, por tanto, el equipo encargado de coordinar la distribución de este archivo, facilitando principalmente información acerca de qué nodos contienen una copia o una parte del archivo buscado.

Recientemente el protocolo BitTorrent contempla otro modo de funcionamiento. Es un modo completamente distribuido que no requiere de la existencia de *trackers*. Se utiliza una tabla de *hash* distribuida DHT para almacenar la correspondencia entre los recursos y los nodos que los comparten. Se podría decir que cada nodo se convierte en parte de un *tracker*, ya que ahora almacenan información que anteriormente correspondía éstos. Este nuevo protocolo se denomina Mainline [230] y se basa en una implementación de Kademlia [231], la DHT más extendida en nuestros días.

A continuación se profundiza en los conceptos básicos sobre DHT, para posteriormente tratar la DHT Kademlia y, por último, detallar brevemente la implementación específica de esta última para el caso de BitTorrent, es decir, Mainline.

6.2.1. DHT

Una tabla *hash* centralizada es una estructura de datos utilizada para almacenar en una misma localización la correspondencia entre valores (*values*) y claves (*keys*).

Una tabla *hash* utiliza una función *hash* sobre una clave, k_1, obteniendo en respuesta un índice id_1 con el que se puede recuperar el valor asociado a la clave, v_1.

En el caso de una DHT esta correspondencia se distribuye entre todas las entidades participantes de la red. Así, cada nodo será responsable de almacenar un conjunto de valores cuyas claves asociadas se encuentren en un rango dado. Una DHT facilita un mecanismo para encontrar al nodo o conjunto de nodos responsables de almacenar un valor utilizando la clave de éste. La metodología utilizada para almacenar de forma distribuida la correspondencia entre clave y valor es la base del diseño de una DHT. Existen múltiples diseños de DHT pero la más utilizada en la actualidad es la denominada Kademlia.

En una DHT, cada participante es identificado por lo que se conoce como identificador de nodo o *ID de nodo*. Tanto los ID de nodos como las claves (también conocidas como *ID de recursos*), suelen ser del mismo tamaño y se derivan del uso de una función *hash* como puede ser MD5, SHA-1, etc. Además, la implementación concreta de la DHT especifica una función de distancia global gracias a la que es posible determinar la distancia entre dos identificadores (ya sean de nodo o de recurso). Esta distancia no está relacionada con la distancia geográfica o el coste computacional para alcanzar a otro nodo, sino que que se relaciona con el espacio de representación utilizado para situar los identificadores.

Cada nodo en la red DHT posee la información de contacto de un conjunto de nodos de la red. Generalmente, la información de contacto de un nodo se refiere a la dirección del nodo en la red que soporta a la DHT (*p.ej.*, dirección IP y número de puerto UDP). Un nodo sólo está conectado a un subgrupo de nodos muy reducido de toda la red DHT. Es común que el tamaño del grupo de nodos al que se conecta sea una función logarítmica del total de nodos en la red, N. Así, si N es 1.000 cada nodo se conectará con unos 7 nodos (si la función es $ln(N)$). Mientras que si N es 1.000.000 el número de nodos al que se conectará cada uno será 14. Es notable que este modelo escala muy bien, permitiendo de este modo la interconexión y compartición entre un número enorme de nodos.

Estas conexiones se escogen de forma estratégica en el espacio en el que se representan los ID de recursos y de nodos, para establecer de la forma más eficiente un mecanismo de enrutamiento. Este mecanismo depende de la implementación concreta de la DHT y es utilizado en la consecución del objetivo principal de la misma, esto es, la búsqueda de identificadores, de nodo o de recursos en la red.

6.2.2. Kademlia

La DHT más utilizada en la actualidad es Kademlia, diseñada por Petar Maymounkov y David Mazìres en [231]. Los nodos en Kademlia son representados como

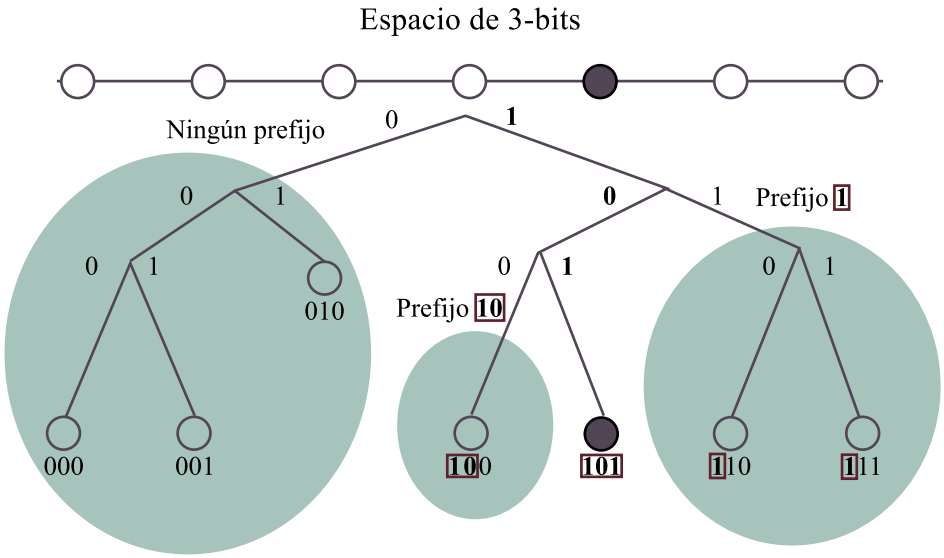


Figura 6.2: Ejemplo de árbol en Kademlia resaltando los *buckets* del nodo 101.

hojas de un árbol binario, cuya posición se determina por el ID de nodo.

La idea principal de Kademlia es que los nodos cuyo ID es cercano a la clave de un recurso son los encargados de almacenar la información relativa a ese recurso. Esta información no es más que la almacenada en la red BitTorrent por los *trackers*, esto es, la lista de nodos que contienen el recurso o parte del recurso identificado por la clave.

Para calcular la distancia entre dos identificadores (ya sean de nodo o de recurso), se utiliza una métrica de distancia: la operación XOR. Nótese que la operación XOR es simétrica, lo que implica que la distancia entre A y B resulta la misma que la distancia entre B y A. Se asume que valores menores de esta métrica implican identificadores más cercanos.

La tabla de rutas de un nodo concreto de Kademlia se organiza en lo que se denominan cubos (*buckets*). Cada uno de estos *bucket* puede contener hasta k nodos que presentan un prefijo de ID de nodo común para los que se almacena la siguiente información: ID de nodo, dirección IP y puerto UDP. El valor concreto de k depende de la implementación particular de Kademlia.

Las tablas de rutas son bastante flexibles ya que un nodo puede escoger los k miembros de cada *bucket* entre un amplio rango de usuarios que contienen el mismo prefijo de ID de nodo común. Por ejemplo, en la Figura 6.2 se muestra un árbol de Kademlia de ejemplo con ID de nodo de 3 bits de longitud. El nodo estudiado en este ejemplo es el que posee el identificador 101, marcado en negro en la figura. Los *buckets* asociados a este nodo están sombreados para identificarlos gráficamente de forma más sencilla. Dado que el árbol está formado por identificadores de longitud 3

bits, cada nodo contendrá en su tabla de rutas 3 *buckets*. Los nodos que se pueden escoger en cada *bucket* se especifican dividiendo el espacio de identificadores en función del número de bits en común con el ID del nodo que está creando su tabla de ruta, es decir, en función del prefijo del identificador de nodo común.

Por ejemplo, para el caso del nodo 101 el primer *bucket* es el correspondiente a todos los nodos que no poseen ningún prefijo en común. Éstos son los que comienzan por 0 y que en la figura se representan como la mitad del árbol que no posee al nodo 101. El segundo *bucket* está formado por todos los nodos que comparten el prefijo 1 y que, por tanto, comienzan por 11 (para que sólo tengan en común el primer 1, marcado con un rectángulo). Éstos son los nodos de la mitad del árbol que comienza por 1 que no posee al nodo. Gráficamente y de forma muy simplificada, de la parte restante del árbol se toma siempre la mitad que no contiene al nodo, y de esta forma se continúa dividiendo el árbol por cada número de bits de prefijo hasta tener un número de divisiones igual al número de bits del árbol. Así se asegura que cada nodo conoce al menos un nodo de cada división, siempre y cuando en esta división exista algún nodo.

6.2.3. Mainline

Como se ha comentado, Mainline es la implementación para el protocolo BitTorrent de Kademlia, que, con más de 5 millones de nodos [232], es actualmente la red DHT más grande con una notable diferencia. Por este motivo ha sido escogida como red de estudio para el presente trabajo.

De la misma forma que Kademlia, su aproximación se basa en que tanto los recursos como los participantes o nodos poseen un identificador de 160 bits. El ID de recurso tiene una longitud de 160 bits y es el resultado de una operación *hash* sobre el recurso en cuestión. El ID de nodo es generado la primera vez que un cliente nuevo inicia la aplicación de BitTorrent correspondiente. Éste será siempre el identificador de este cliente a menos que la aplicación de BitTorrent sea completamente desinstalada o se elimine su fichero de preferencias. Así que podemos suponer el ID de nodo como un identificador único por cliente incluso en la circunstancia de que este cliente cambie de dirección IP.

Como una implementación de Kademlia que es, la mayoría de los aspectos de diseño son comunes a ésta. Sin embargo, existen algunos detalles que se añaden en la implementación de Mainline. Los principales son los tres siguientes:

1. La clave tiene una longitud de 160 bits y se obtiene a partir del algoritmo SHA-1 del recurso que será identificado por la clave. Se utiliza el algoritmo SHA-1 para reducir las posibilidades de que aparezcan dos recursos con una misma clave. Además, el uso de una función *hash* garantiza la distribución uniforme

en el árbol de los recursos. El valor del *hash* se almacena en el archivo `.torrent` correspondiente.

2. Se escoge un valor para k de 8 nodos, lo que implica que el número máximo de nodos almacenados en cada *bucket* es 8.

3. El tiempo en el que un nodo permanece en la lista de nodos que poseen un recurso son 30 minutos. Después de este tiempo, o el nodo reenvía otro mensaje indicando que aún comparte el recurso correspondiente, o se elimina al nodo de la lista de nodos que comparten el recurso.

En la DHT Mainline los valores son listas de nodos que comparten actualmente el recurso identificado por la clave asociada. Así, cualquier búsqueda en esta red da como resultado una lista de nodos identificados por la dirección IP y el puerto TCP del protocolo BitTorrent. Una vez un nodo de la red obtiene esta lista, es capaz, por medio del protocolo BitTorrent, de conectarse a uno o varios nodos de la red, y comenzar el proceso de descarga y compartición del recurso buscado.

Nótese que aunque las listas de nodos son los valores de la DHT, éstos son dinámicos, ya que crecen y decrecen en función de los mensajes de los nodos indicando que comienzan a compartir un recurso o que han dejado de hacerlo.

Principales mensajes de Mainline

A continuación presentamos los principales mensajes en Mainline que son una adaptación de los originales mensajes de Kademlia: `ping`, `find_node`, `find_value` y `store`. Para una información más detallada puede consultarse [230]:

- `ping`

  Comprueba si un nodo continúa estando activo. El nodo receptor del mensaje también aprende la existencia del nodo que originó el mensaje.

- `find_node`

  El nodo que recibe este mensaje envía en respuesta los k nodos más cercanos en su tabla de rutas al identificador ID_n solicitado.

- `get_peers`

  Pide información sobre un archivo f con identificador de archivo ID_f. Este mensaje sigue un procedimiento que funciona de forma iterativa. En cada iteración del proceso, cada salto, los nodos responden al mensaje con las direcciones IP, puertos e identificadores de nodo de los nodos más cercanos a ID_f. En el último salto del proceso los nodos devuelven una lista con las direcciones IP,

puertos e identificadores de nodo de los nodos (N_f) que poseen una copia del recurso buscado, f.

- `announce_peer`

  Un nodo anuncia con este mensaje que posee un archivo (o una parte de él) con un identificador de recurso ID_f, y como consecuencia de este anuncio comienza a formar parte de la lista de nodos que lo comparten (valor asociado a la clave del recurso en la DHT, N_f). Un nodo envía estos mensajes a los k nodos con identificadores más cercanos a ID_f. Estos k nodos más cercanos han sido previamente encontrados con un mensaje del tipo `get_peers`. El nodo que envía este mensaje es responsable de reenviarlo cada cierto periodo de tiempo para evitar que expire su entrada en la lista de nodos que poseen el archivo.

Proceso de compartición de un recurso en la red Mainline

Finalmente, el procedimiento más interesante de la red Mainline para el presente trabajo es el de publicación de un recurso compartido que se lleva a cabo mediante los mensajes de tipo `announce_peer`. El proceso puede resumirse en los siguientes pasos (ver Figura 6.3):

1. *Inicio de la compartición*. El inicio de la compartición de un recurso puede comenzar por dos motivos. Primero, un nodo de la red puede generar un recurso y el archivo `.torrent` asociado (con el *hash* que identifica al recurso) y, posteriormente, subirlo a un servidor web especializado para que los usuarios interesados puedan encontrarlo. Segundo, el proceso más común, un usuario inicia su aplicación de BitTorrent y comienza la compartición de todos los recursos que tenga en su carpeta de recursos compartidos (generalmente los que descargó con anterioridad de la red).

2. *Búsqueda de* peers. En segundo lugar, es necesario encontrar a los nodos que tienen la responsabilidad de almacenar la información relativa al recurso a compartir. Estos nodos son aquellos cuyo identificador de nodo es cercano al identificador de recurso (ID_f en el ejemplo). Para encontrarlos se envía un mensaje del tipo `get_peers`. En respuesta a este mensaje se obtienen en sucesivas aproximaciones una lista de nodos (N_ID) cuyo identificador de nodo es cercano al identificador de recurso que se indicó en el mensaje `get_peers` y finalmente, una lista de nodos que poseen el recurso ID_f o parte de él, ésta es N_f.

3. *Anuncio de* peers. Por último, un mensaje del tipo `announce_peer` es enviado por el cliente identificado por $(ID_x)^n$ a la lista de nodos con un ID cercano al recurso ID_f (N_{ID} en la figura). Como consecuencia del envío de este mensaje, el

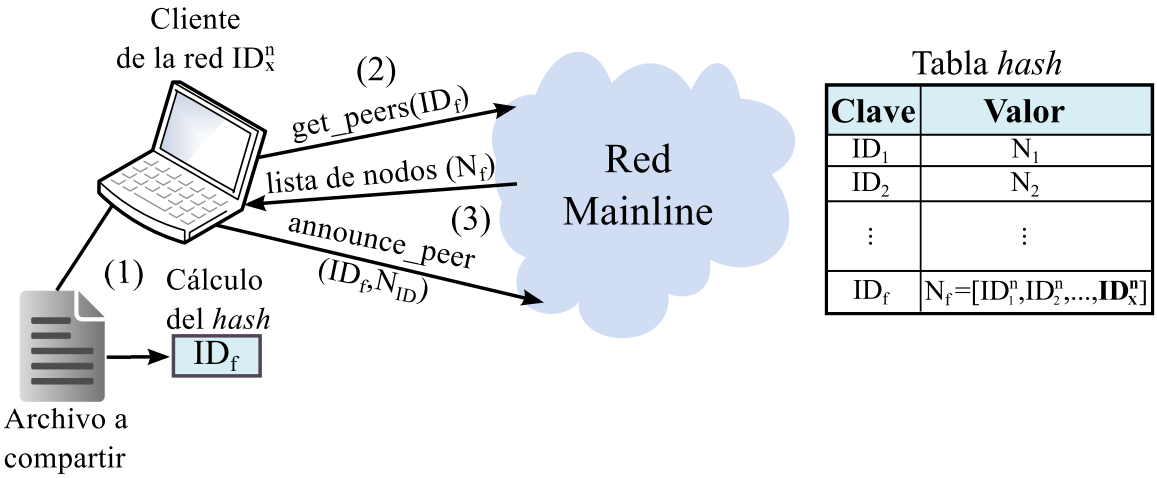


Figura 6.3: Esquema del proceso seguido por un nodo de la red Mainline para almacenar en la red la información de que él comparte un recurso dado: (1) inicio de la compartición, (2) búsqueda de *peers* y (3) anuncio de *peers*.

identificador $(ID_x)^n$ es incluido como un nodo más en la lista de nodos asociada a la entrada de la tabla *hash* identificada por la clave ID_f.

Nótese que todos los nodos que posean una copia o parte de una copia de un recurso concreto, notificarán al mismo conjunto de nodos que poseen el mencionado recurso (enviando un mensaje del tipo `announce_peer`). Este conjunto de nodos son aquellos que presenten un identificador de nodo más cercano al del recurso compartido. Así, la misma zona del árbol binario de Mainline recibirá siempre todos los mensajes `announce_peer` asociados con los mismos recursos.

6.3. Compartición de recursos en redes P2P: Modelo de comportamiento

La hipótesis que sustenta nuestra propuesta de detección de *botnets* P2P parásitas es que los recursos compartidos por usuarios legítimos de redes P2P (*recursos legítimos*) serán accedidos de una forma diferente a la de los recursos compartidos por nodos pertenecientes a una *botnet* parásita que utilice la red (*recursos* botnet). De este modo, nuestro objetivo es, en primer lugar, establecer un modelo de comportamiento en la compartición de recursos para cada caso (P2P *vs.* P2P parásita). A partir de ello, será posible posteriormente llevar a cabo el diseño y puesta en marcha de un sistema de detección de *botnets* parásitas sustentado sobre dicho modelado. Nótese que, por simplicidad, de ahora en adelante cuando nos refiramos a una *botnet* nos referiremos

realmente a las *botnets* objeto de estudio de este capítulo, esto es, las *botnets* P2P parásitas.

En nuestro caso vamos a basar los modelos de compartición de recursos legítimos o de recursos *botnets* en la evolución temporal del *número de nodos P2P que los comparten*. Así, definimos $n_r(k)$ como el número de nodos que comparten un recurso r durante un periodo de tiempo de duración δ que finaliza en $k \cdot \delta,\ k = 1,...,K$.

En la Sección 6.4 se detallará una metodología para estimar $n_r(k)$. Por medio de ella ha sido posible monitorizar en la red Mainline la evolución de la compartición de 71.135 recursos durante 3 meses (ver detalles en la Sección 6.6), evidenciándose que existen patrones de evolución de $n_r(k)$ diferentes. Entre los resultados hallados se constata que existen recursos que, al menos durante un período de tiempo, son compartidos por un elevado número de usuarios. Estos recursos son denominados *recursos populares*. Definimos así un umbral de popularidad, θ_P, de modo que un recurso dado será considerado como popular sí y sólo sí existe un valor k, con $k \epsilon [1, K]$, que cumple $n_r(k) > \theta_P$. En la Sección 6.5 se especificará cómo hallar el umbral θ_P.

En nuestro modelado de compartición de recursos asumimos dos fases fundamentales en la evolución de todo recurso popular: *fase de compartición* y *fase de desaparición*. En la fase de *compartición* el recurso comienza a ser descargado y crece en popularidad, alcanzando un periodo en el que el recurso es compartido por un número elevado de nodos. Tras esto, el interés en el recurso decrece rápidamente hasta alcanzar un punto desde el que la disminución del número de nodos que lo comparten se hace más suave. La fase de *desaparición* va desde este punto, en el que el decrecimiento es menos acusado, hasta que el recurso no es compartido por ningún usuario y desaparece de la red.

En la Figura 6.4 se muestra la evolución de $n_r(k)$ para un recurso P2P legítimo típico. En esta figura se observa que la evolución de $n_r(k)$ presenta claramente las dos fases descritas con anterioridad. La fase de compartición va desde el inicio de la compartición hasta la hora 100, aproximadamente. La fase de desaparición, por su parte, comienza en la hora 100 y termina en algún punto tras la hora 500 (la desaparición completa no aparece en la figura).

Tomando como base el comportamiento usual antes comentado para recursos P2P populares legítimos, la hipótesis principal de detección de nuestro sistema es que el comportamiento de $n_r(k)$ de los recursos legítimos y populares compartidos en redes P2P será sustancialmente distinto con respecto a los patrones de $n_r(k)$ cuando r representa a un recurso *botnet*. Si esta hipótesis es cierta, un sistema de detección de recursos *botnet* podría basarse en la detección de desviaciones con respecto al comportamiento del $n_r(k)$ esperado para recursos legítimos.

El problema principal para verificar esta hipótesis es que no es posible utilizar trazas reales de recursos pertenecientes a *botnet* parásitas en la fase de experimentación.

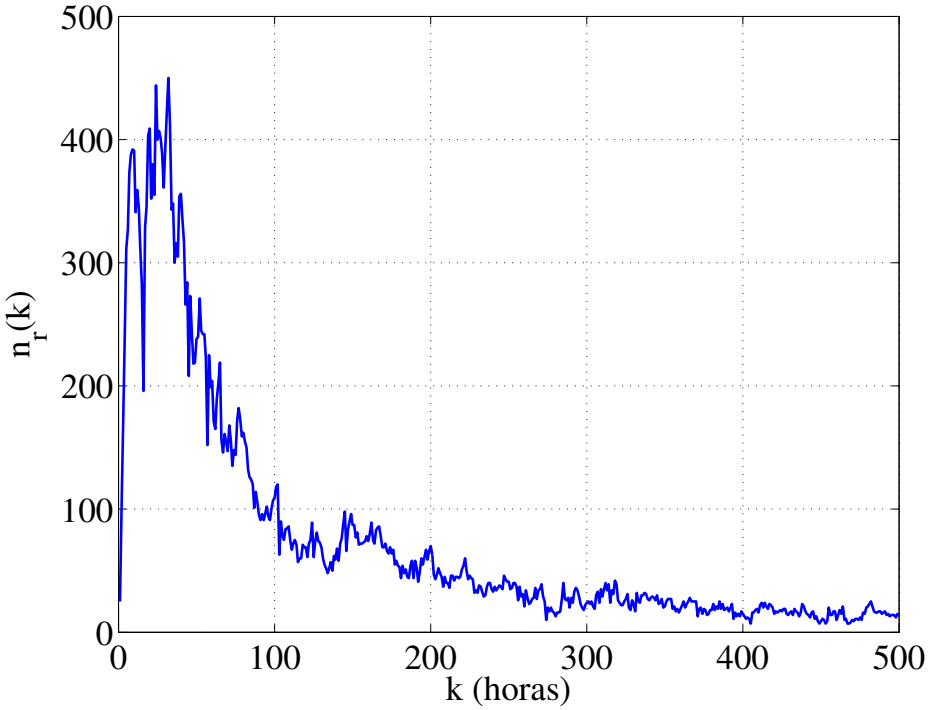


Figura 6.4: Evolución de la compartición, $n_r(k)$ para un típico recurso P2P legítimo (δ=1 hora).

Esto se debe a que, hasta donde alcanza el conocimiento de los autores, hoy en día no existen *botnets* parásitas activas que hayan sido reportadas, posiblemente debido a que es extremadamente complicado detectarlas en base a los métodos existentes. Por esta razón, la estrategia que se presenta en este capítulo consiste en construir un modelo teórico para la evolución temporal de compartición de los recursos *botnet*, asumiendo que los *botmasters* deben cumplir ciertas reglas para mantener el control de sus *botnets*.

A continuación se indican dos características en las que se basa el modelo aquí asumido para recursos *botnet*. Estas características representan las restricciones que no pueden ser evadidas por los *botmasters* sin degradar el comportamiento de sus *botnets* o exponerlas a los mecanismos actuales de detección:

1. **Los recursos *botnets* deben ser recursos populares**

   Las *botnets* están compuestas por un gran número habitual observado de *bots* [136]. Cuando los *botmasters* actualizan el código del *bot* o envían comandos, todos los *bots* deben descargar el recurso *botnet* que contiene los mencionados comandos o actualizaciones. Esto implica que cada interacción entre el *botmaster* y sus *bots* repercute en un enorme número de descargas del recurso *botnet* asociado a estas comunicaciones.

Por lo anterior, en una primera fase de nuestro estudio, no se considerarán *botnets* de un tamaño reducido, ya que su amenaza se considera poco significativa. Como se estudiará en la Sección 6.6, el tamaño mínimo de una *botnet* para ser analizada en la fase experimental es de 1.000 nodos, asunción bastante relajada ya que el número de *bots* de una *botnet* supera ampliamente este umbral.

A pesar de lo antes expuesto, es importante resaltar en este punto que la condición de que un recurso *botnet* sea popular no es estrictamente necesaria ya que un *botmaster* podría evadirla creando múltiples recursos para grupos reducidos de *bots*, en lugar de un único recurso para toda la *botnet*. Sin embargo, el incremento en el número de nuevos recursos que aparecen en la *botnet* sería también una alarma en base a la que construir un sistema de detección. Esta condición alternativa al alto número de nodos compartiendo un recurso constituye una línea de trabajo futuro.

2. **Los recursos *botnets* deben presentar un tiempo de compartición breve**

   El periodo de tiempo durante el que un recurso *botnet* es compartido debe ser reducido, pues este archivo apunta directamente a todos los miembros de la *botnet* y los expone al riesgo de ser detectados. Así, conociendo un solo recurso *botnet* sería sencillo extraer todos los miembros de la *botnet* sin más que buscar los nodos que comparten este recurso.

Estas dos características implican que, durante la *fase de compartición* de un recurso *botnet*, todos los *bots* que lo descargan también lo comparten para asegurarse que éste sea accedido por el mayor número de *bots* de la *botnet*. Así, durante esta fase es esperable una alta popularidad del recurso.

Por otro lado, y a fin de minimizar la probabilidad de ser descubiertos por los mecanismos de detección, se espera que en la *fase de desaparición* los *bots* dejen de compartir el recurso *botnet* en un corto periodo de tiempo.

La evolución concreta de estas fases dependerá notablemente de múltiples factores, de modo que el modelo teórico propuesto considera específicamente los siguientes:

- *Número de* bots *de una* botnet *P2P parásita*, N

  Como se ha mencionado anteriormente, consideramos valores elevados para N, ya que de otra forma la *botnet* no supondría una amenaza relevante. Los valores considerados para N en el entorno experimental desarrollado se describen en la Sección 6.6.

- *Duración media de la fase de compartición de los recursos* botnet, t_{sh}

  Tras la discusión anterior, este valor se espera que sea bajo, dado que al *botmaster* le interesa reducir la probabilidad de ser detectado por parte de los agentes de

seguridad. Esto es, hay que minimizar el tiempo durante el que existen recursos que apuntan directamente a los miembros de la *botnet*.

- *Tasa media de llegada y salida de los* bots, λ

  Como apuntan algunos trabajos [233], la tasa de llegada y salida de los miembros de una *botnet* sigue una distribución de Poisson, en la que λ representa la media de estas distribuciones.

  Según lo establecido con anterioridad, la transición de la fase de compartición a la fase de desaparición se espera brusca, o al menos más brusca que para el caso de la evolución de la compartición de un recurso legítimo.

- *Intervalo de desaparición*

  La orden del *botmaster* indicando a sus *bots* que han de dejar de compartir un recurso concreto debe distribuirse por toda la *botnet*. Dadas las desviaciones en la sincronización de los *bots* entre otros factores, es de esperar la existencia de un reducido intervalo de desaparición que parta de la situación de un número elevado de *bots* compartiendo el recurso para desembocar en la situación de que todos los *bots* han dejado de compartirlo. Este intervalo es modelado por medio de una distribución uniforme $U[0, \Delta \cdot t_{sh}]$.

Siguiendo este modelo teórico, en la Figura 6.5 se muestra la evolución de $n_r(k)$ para algunos ejemplos de recursos *botnet* sintéticos con diferentes parámetros. Nótese que existen dos diferencias fundamentales con respecto a la evolución de $n_r(k)$ para un recurso legítimo y para un recurso *botnet* (compárense las Figuras 6.4 y 6.5). La primera aparece alrededor del instante t_{sh}, donde $n_r(k)$ cae abruptamente. La segunda es la reducida duración de la compartición del recurso *botnet* entre 1 y 3 días y tras esta escasa compartición el recurso desaparece completamente de la red. Estas dos diferencias son las razones que han derivado en la construcción de un sistema de detección de recursos *botnets* basado en la monitorización de $n_r(k)$. El algoritmo de monitorización de la red Mainline utilizado se describe en la siguiente sección.

Adicionalmente, es de resaltar que los comandos del *botmaster* son muy frecuentes y un *bot* debe intentar recibir todos ellos. Esto implica que el *botmaster* compartirá un gran número de recursos con todos los miembros de su *botnet*, con lo que existirán múltiples oportunidades para que el sistema de detección propuesto pueda alertar de la existencia de estos recursos anómalos. Este hecho facilita enormemente la tarea de detección al hacer menos crítica la tasa de aciertos del sistema, ya que con una tasa de aciertos relativamente baja, el sistema propuesto sería capaz de extraer algunos de los recursos compartidos por la *botnet* y de ellos la mayoría de los *bots* activos.

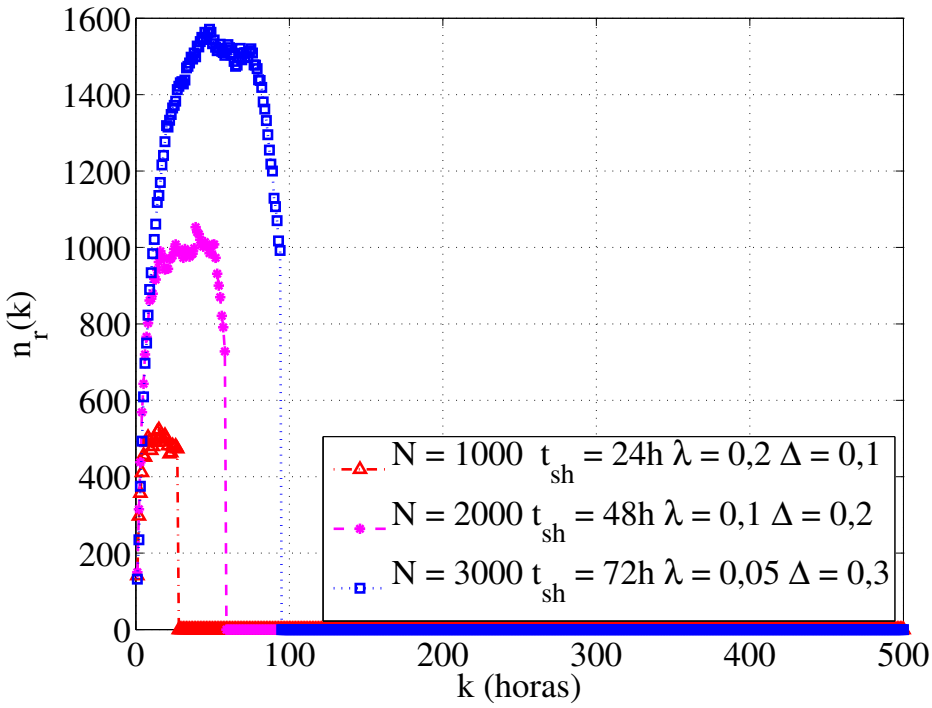


Figura 6.5: Evolución temporal de diferentes actualizaciones sintéticas de una *botnet* parásita (δ= 1 hora).

6.4. Monitorización de los recursos compartidos en una red P2P

En la sección anterior se describe la base metodológica teórica que sustenta el comportamiento normal de la compartición de recursos en redes P2P. Tomando ello como ejes y aceptando el comportamiento ya descrito para la compartición de recursos *botnets*, se pretende detectar comportamientos anómalos posteriores. Para esto, es preciso un proceso previo de monitorización a fin de determinar la evolución temporal de los recursos compartidos en la red P2P objeto de estudio, es decir $n_r(k)$. Este proceso de monitorización es descrito en lo que sigue, considerándose la red BitTorrent por ser la red de compartición de recursos mediante P2P más utilizada en la actualidad [234]. Concretamente, esta monitorización se realiza en la red Mainline, implementación de Kademlia para el protocolo BitTorrent que evita la necesidad del uso de *trackers*.

El proceso de monitorización de los recursos de la red Mainline se compone de dos módulos principales: (*i*) *crawling* de la red y (*ii*) *sniffing* de los mensajes de una zona. A continuación se detallan ambos módulos.

6.4.1. *Crawling*

Este módulo se basa en el trabajo realizado en [235] (*Blizzard*) y posteriormente adaptado a Mainline en [234]. Llamaremos a este módulo *Blizzard2* por ser la modificación de *Blizzard*.

El objetivo fundamental de *Blizzard2* es extraer una lista de los nodos activos en Mainline (realizar un *crawling*) con una alta frecuencia. En suma, como resultado del proceso de *crawling* se obtiene una lista de nodos altamente activos, y para cada uno de ellos: el ID de nodo, la dirección IP y el puerto, y el instante temporal en el que el nodo respondió al *crawler*.

El problema principal de realizar el *crawling* de una DHT grande, como es el caso de Mainline, es que el *churn* de los nodos es muy elevado y, por tanto, los nodos activos en la red varían con una gran frecuencia. Por este motivo, es preciso que el tiempo empleado en realizar el *crawling* sea lo menor posible para obtener una "foto" real de la red. Con el fin de acelerar este proceso, otros sistemas (como [236]) se ejecutan en paralelo en múltiples máquinas. Sin embargo, en la paralelización del proceso de *crawling*, gran parte del tiempo de CPU se emplea en tareas de sincronización entre distintas máquinas. Así que para agilizar todo lo posible la extracción de nodos de la red, *Blizzard2* es ejecutado en una misma máquina de modo que toda la información recogida se recibe directamente en la máquina que lanza el *crawler*.

La estructura de ejecución de *Blizzard2* se puede resumir en los siguientes pasos:

1. Comienza contactando con uno o varios nodos de tipo semilla que son introducidos manualmente en el sistema. Este proceso es muy similar al registro que utilizan los clientes de BitTorrent que se conectan a la red Mainline.

2. Posteriormente se pregunta al nodo o nodos semilla por un conjunto de nodos con los que comenzar el proceso de descubrimiento.

3. A cada nuevo nodo descubierto se le envía un mensaje para descubrir nuevos nodos. De la respuesta obtenida (lista de nodos) se extraen los nodos aún desconocidos y se incluyen en la lista de nodos descubiertos por el *crawling*.

4. Toda la información anterior se almacena en memoria principal para agilizar el proceso. Una vez terminado el *crawling*, la lista completa de nodos descubiertos es almacenada en disco.

Al inicio del proceso de *crawling* el número de nodos desconocidos que son descubiertos crece de forma exponencial. Posteriormente, el número de nodos descubiertos se acerca asintóticamente al número total de nodos de la red. El criterio de parada de este proceso es delicado y si se continúa de forma indefinida los datos recogidos no

Algoritmo 3 Hebra de envío de los mensajes de descubrimiento.

```
1: lista_nodos.add(semillas);
2: n_veces = 16;
3: mientras posicion < 0,99 * longitud(lista_nodos) hacer
4:    para i = 0 hasta 16 hacer
5:       nodos_destino = lista_nodos[posicion];
6:       enviar(find_node, nodos_destino, n_veces);
7:       posicion++;
8:    fin para
9: fin mientras
```

Algoritmo 4 Hebra que recibe las respuestas de los mensajes de descubrimiento.

```
1: lista_nodos; //Dato compartido entre ambas hebras.
2: mientras verdad hacer
3:    mensaje = escucha(mensajes == respuesta a find_node);
4:    para nodo en mensaje hacer
5:       si nodo no está en lista_nodos entonces
6:          lista_nodos.add(nodo);
7:       fin si
8:    fin para
9: fin mientras
```

serán precisos, ya que existirá un número importante de nodos que han dejado la red. Como el tiempo total en el que se completa el *crawling* es más importante que el descubrimiento de la totalidad de los nodos de la red, los autores de *Blizzard* detienen el envío de mensajes de descubrimiento cuando se han enviado estos mensajes a un 99% de los nodos descubiertos (así se hace también en *Blizzard2*). Adicionalmente, tras la detención en el envío de mensajes de descubrimiento para recoger las respuestas a los mensajes enviados con anterioridad el *crawler* permanece a la escucha durante 30 segundos extra.

La implementación del *crawler* se distribuye en dos hebras asíncronas: (*i*) una hebra para enviar los mensajes del tipo `find_node` (Algoritmo 3) y (*ii*) otra para recibir e interpretar las respuestas a los mensajes enviados por la primera hebra (Algoritmo 4).

Existe en ambos algoritmos una lista compartida en la que se almacenan los nodos descubiertos durante el proceso de *crawling* (*lista_nodos*). La hebra receptora añade a la lista de nodos aquellos que aún no están en ella y que son extraídos de las respuestas a los mensajes `find_node`.

Por su parte, la hebra de envío manda 16 mensajes (*n_veces*) del tipo `find_node` a cada nodo de la lista compartida. El valor del ID de nodo origen de cada uno de esos

16 mensajes es cuidadosamente escogidos para estar situado en *buckets* diferentes y así minimizar el solapamiento de la lista de nodos obtenidos como respuesta.

Finalmente, cuando han sido contactados un 99 % de los nodos en $lista\_nodos$ el *crawling* finaliza, deteniéndose la hebra de envío y, tras 30 segundos, la de recepción, que escribe la lista de nodos descubiertos a un archivo en disco.

Crawling de una zona

Un *crawling* completo de la red Mainline genera una gran cantidad de tráfico y es extremadamente costoso (con picos de descarga de 100 Mbit/s). Además, es necesario realizarlo con bastante frecuencia para disponer de una lista de nodos activos de la red lo más actual posible. Por esta razón, se puede realizar el *crawling* de una zona concreta de la red Mainline con un prefijo de 8-bits en común. Así, el *crawler* tiene como objetivo extraer los nodos activos en la red Mainline que comparten un mismo prefijo de 8 bits de longitud. Esto supone una de las 256 partes (2^8) en las que se divide toda la red Mainline lo que implica un porcentaje bastante representativo de la misma. Este *crawling* es realizado en menos de 2,5 segundos.

6.4.2. *Sniffing*

Este módulo se basa en el trabajo de [234], el cual ha sido adaptado a nuestro caso particular. El objetivo es recopilar los nodos que en cada instante de la monitorización comparten alguno de los recursos asociados a una zona concreta de la red Mainline. Para esto se incluyen en la DHT una gran cantidad de nodos *sybil* con identificadores de nodo cercanos a todos los nodos activos en esa zona (esta información es facilitada por el módulo anterior). Así, los *sybils* creados serán vecinos de todos los nodos de la zona y serán capaces de escuchar los mensajes que se dirijan a ésta.

Sea n un nodo de la zona a monitorizar e ID_n su identificador de nodo. Este módulo inserta 256 *sybils* cuyos identificadores de nodo coinciden en los primeros 47 bits de ID_n. Esto hace que la probabilidad de que exista un nodo en la DHT de Mainline con un identificador más cercano a ID_n sea extremadamente reducida. Los identificadores de nodo de los *sybils* se generan variando los bits en el intervalo 48-55 (8 bits) y escogiendo de forma aleatoria los bits restantes (56-160), como resultado, se obtienen 256 identificadores de nodo diferentes pero muy cercanos al identificador de nodo ID_n.

El proceso de inclusión de los *sybils* como nodos de la DHT se basa en notificar su existencia por mensajes tipo `ping` con identificadores de nodo origen diferentes, uno por cada *sybil*. Estos `pings` son enviados a los nodos reales con ID de nodo cercano al objetivo n. Los nodos reales como consecuencia incluirán a nuestros *sybils* dentro de

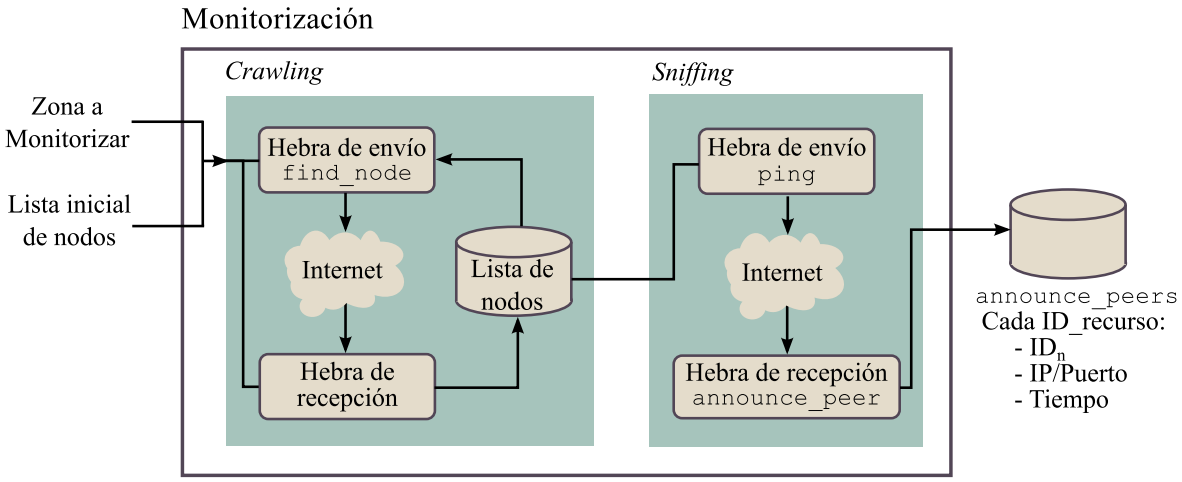


Figura 6.6: Esquema de la arquitectura del sistema de monitorización de la red Mainline con sus dos componentes fundamentales: *crawling* y *sniffing*.

sus tablas de rutas y, así posteriormente, propagarán la existencia de éstos a otros nodos en la DHT.

En resumen, el proceso de monitorización de recursos funciona como sigue (ver Figura 6.6). Primero, se descubren los nodos dentro de la zona a monitorizar utilizando el módulo de *crawling*. Tras esto, se envían mensajes del tipo `ping` con identificadores de nodo fuente muy cercanos a todos los nodos activos en dicha zona, notificándose la existencia de los *sybils* creados y realizando, de esta forma, un ataque de polución a las tablas de ruta de los nodos reales de la zona. Por medio de este ataque se insertan los *sybils* como vecinos de los nodos a monitorizar. Hecho esto, para alcanzar a cualquier nodo de la zona monitorizada se enviará primero una solicitud a uno de nuestros *sybils* ya que son vecinos muy cercanos a los nodos objetivo.

Segundo, se almacena la información de los mensajes `announce_peers`, que son los que apuntan a los nodos que poseen realmente los recursos que se pretenden monitorizar. Para cada mensaje `announce_peers` se almacena el ID del recurso anunciado, la IP y puerto de escucha del nodo origen del mensaje, el ID del nodo y una marca temporal del instante en el que llegó el mensaje. Como resultado de este proceso, es posible calcular $n_r(k)$ para cada recurso monitorizado r y un conjunto de intervalos de duración δ dado.

Finalmente, existe una limitación en cuanto a la proporción de la red Mainline que el sistema es capaz de monitorizar. Es posible que el módulo de *crawling* realice un *crawling* completo de la red, aunque éste tome más tiempo y suponga un consumo de ancho de banda muy elevado. Sin embargo, desde el módulo de *sniffing* es extremadamente complejo disponer del ancho de banda necesario para insertar múltiples

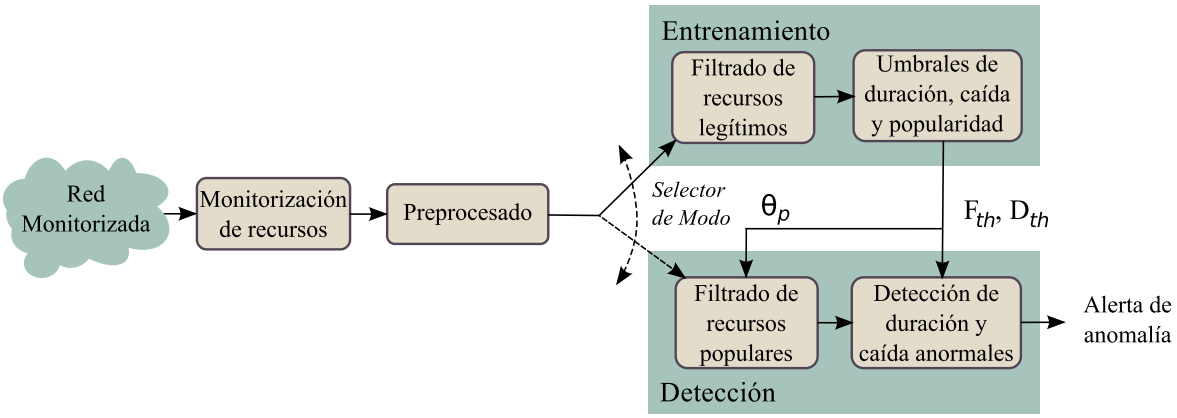


Figura 6.7: Arquitectura funcional del sistema de detección propuesto.

sybils por cada uno de los miembros de la red Mainline y que, adicionalmente, estos nodos *sybils* envíen periódicamente mensajes a sus vecinos. Esta limitación no ha de afectar al sistema que utilice posteriormente la monitorización, ya que sí es posible realizar una monitorización de los mensajes intercambiados en una zona de 1/256 del total de la red Mainline, lo cual es bastante representativo del comportamiento general de la red.

6.5. Sistema para la detección de *botnets* P2P parásitas

En base a los modelos de comportamiento de compartición presentados en la Sección 6.3, se pretende construir un sistema de detección capaz de monitorizar $n_r(k)$ para los diferentes recursos de una red P2P y de detectar potenciales recursos *botnet*.

La arquitectura de este sistema de detección basado en la evolución de los recursos compartidos se muestra en la Figura 6.7. Es de destacar que los datos extraídos por medio del módulo de monitorización de recursos (descrito en la Sección 6.4) constituyen la entrada u observaciones a evaluar. Como en todo sistema de detección en éste también se consideran las tres etapas típicas: (*i*) preprocesado, (*ii*) entrenamiento y (*iii*) detección.

- *Entrenamiento*

  Primero, se construye un modelo de normalidad a fin de representar la evolución usual de la compartición de recursos legítimos en la red P2P monitorizada.

- *Detección*

  Una vez que el modelo anterior ha sido obtenido, cada recurso posterior compartido en la red P2P es analizado a lo largo de su evolución temporal con el

objetivo de determinar posibles desviaciones con respecto al comportamiento esperado. En caso de que se produzca dicha desviación, se lanza una alarma que indica la detección de un recurso potencialmente perteneciente a una *botnet*.

- *Preprocesado*

  Ambas etapas, tanto entrenamiento como detección, precisan de una etapa previa común de preprocesado. Esta etapa procesa la información recogida por el módulo de monitorización para adaptarla a nuestros intereses y así utilizarla en los módulos de detección o entrenamiento.

En lo que sigue se discute cada una de las etapas anteriores.

6.5.1. Preprocesado

Como primer paso del preprocesado derivado del proceso de *crawling* explicado en la Sección 6.4.1 cada recurso monitorizado r es representado por un vector temporal $\boldsymbol{n_r}$:

$$\boldsymbol{n_r} = [n_r(1), n_r(2), ..., n_r(K)] \tag{6.1}$$

donde $n_r(k)$ representa el número de nodos que comparten un recurso r en el período número k de duración δ, durante un intervalo de monitorización $T = K \cdot \delta$. Cada muestra $n_r(k)$ recibida desde el módulo de monitorización se añade a $\boldsymbol{n_r}$, de modo que $\boldsymbol{n_r}$ representa realmente la evolución temporal del número de nodos que comparten el recurso r. Diremos que un vector $\boldsymbol{n_r}$ está *completo* cuando $n_r(k) = 0, k > K$.

Tras construir $\boldsymbol{n_r}$, se realiza un filtrado paso baja a fin de reducir los comportamientos espúreos de la evolución temporal. Estos espúreos se deben principalmente a la elevada tasa de entrada y salida de los nodos (*churn*) propia de las redes P2P. Por ejemplo, si un recurso monitorizado r es compartido por nodos de la misma zona geográfica, $\boldsymbol{n_r}$ exhibirá muy probablemente un comportamiento periódico alrededor de 24 horas debido a las desconexiones de los equipos durante la noche.

Por cada punto en la serie temporal $\boldsymbol{n_r}$, el filtrado se realiza tomando el máximo valor dentro de una ventana de tamaño W muestras (siendo W un número impar). La ventana resultante se compone de un número de intervalos de duración δ centrados en el intervalo k. Así, se obtiene una nueva serie temporal $\hat{\boldsymbol{n}}_{\boldsymbol{r}} = [\hat{n}_r(1), \hat{n}_r(2), ..., \hat{n}_r(K)]$ donde

$$\hat{n}_r(k) = max_i\Big\{n_r(i)\Big\}, \;\; i = k - \frac{W-1}{2}, ..., k + \frac{W-1}{2} \tag{6.2}$$

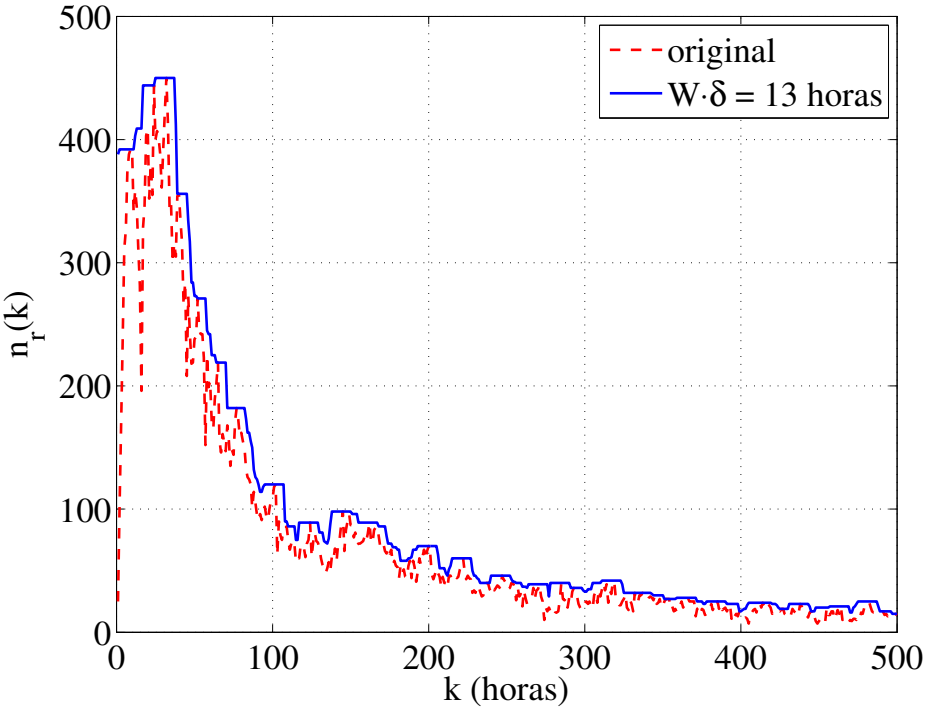


Figura 6.8: Evolución temporal de un recurso y su correspondiente filtrado de máximo (δ=1 hora).

En la Figura 6.8 se muestra el resultado del filtrado de máximos de un recurso popular monitorizado para $W = 13$. Como se esperaba, la aplicación del filtro suaviza la evolución temporal disminuyendo los comportamientos espúreos; lo que en suma repercutirá en una disminución de los errores de las etapas sucesivas.

6.5.2. Entrenamiento

El objetivo principal de este módulo es hallar un modelo de normalidad asociado a la evolución temporal de los recursos compartidos en la red P2P monitorizada. Para esto es necesario reparar en los siguientes aspectos.

Base de datos de recursos

Primero se debe recopilar una base de datos con la evolución de todos los recursos compartidos en la red P2P monitorizada, cada uno representado por un vector $\hat{\boldsymbol{n}}_r$ *completo*.

Filtrado de recursos legítimos

Dado que en la fase de entrenamiento el objetivo es modelar el comportamiento de recursos legítimos, se deben recopilar solamente este tipo de recursos. De otra forma, se podrían derivar modelos incorrectos de normalidad.

Para este propósito se pueden seguir diferentes aproximaciones. La aproximación más directa pero a la par más complicada sería descargar el recurso y comprobar manualmente si el contenido compartido es legítimo o no. Se sigue, sin embargo, una estrategia automática a través de la que se extrae cierta información de los recursos monitorizados consultando páginas web especializadas, como son: `http://btdigg.org/` o `http://torrentz.eu/`. Se podría asumir que la existencia de información acerca de estos recursos en páginas web públicas de búsqueda de archivos `.torrent` es una prueba de que estos recursos son legítimos y no pertenecen a una *botnet*.

Umbral de caída

Estamos interesados en identificar aquellos recursos que exhiban una caída abrupta en el número de nodos que los comparten, ya que esto es un comportamiento esperable en los recursos *botnet*. Así, se calculará para cada recurso r la máxima diferencia entre dos valores consecutivos de $\hat{\boldsymbol{n}}_{\boldsymbol{r}}$, lo que denotamos como F_r:

$$F_r = |min_k\{\hat{n}_r(k) - \hat{n}_r(k+1)\}|; \;\; k = 1, ..., K-1 \tag{6.3}$$

Después, el umbral de caída, F_{th}, se obtiene como un factor α sobre la máxima caída de todos los recursos monitorizados en la fase de entrenamiento:

$$F_{th} = \alpha \cdot max_r\{F_r\}; \;\; \forall r \tag{6.4}$$

Nótese que el valor α modificará la sensibilidad del sistema de detección. Un valor de α bajo implicará una tasa de falsos positivos elevada, mientras que valores demasiado altos de α repercutirán en bajas tasas de detección. En la Sección 6.6 se muestra la curva ROC del sistema variando este parámetro.

Umbral de popularidad

Como se ha indicado previamente, sólo los recursos populares son considerados en este sistema de detección como posibles recursos *botnet*, ya que se asume como una restricción necesaria que estos recursos sean compartidos por un elevado número de *bots*, es decir, que sean populares.

La selección del valor del umbral de popularidad, θ_P, podría ser una tarea realmente compleja en una aplicación genérica. Afortunadamente, en el caso presente esto es muy diferente ya que el sistema de detección se basa en caídas mayores de F_{th}, lo que implica necesariamente que al menos ha de existir un valor k del recurso r para el cual $\hat{n}_r(k) \geq F_{th}$. Por este motivo, en nuestro caso tomamos $\theta_P = F_{th}$ y filtramos aquellos recursos que no cumplan la condición

$$max_k\{\hat{n}_r(k)\} \geq \theta_P; \;\; k = 1, ..., K \tag{6.5}$$

Umbral de duración

La segunda restricción que han de cumplir los recursos *botnets* es que se espera que su tiempo de vida sea reducido, para evitar que existan recursos en la red que apunten durante un elevado tiempo a todos los miembros de la *botnet*. El tiempo de vida, T_r, puede ser definido como el tiempo que pasa desde que un recurso r dado aparece en la red hasta que desaparece completamente de la misma (dejan de observarse mensajes `announce_peers` para dicho recurso). Así, de forma análoga a como sucede con el umbral de caída, el umbral de duración, D_{th}, se puede definir como un factor del T_r mínimo observado en el entrenamiento:

$$D_{th} = \beta \cdot min_r\{T_r\}; \;\; \forall r \tag{6.6}$$

6.5.3. Detección

La última etapa del sistema de detección consiste en determinar la naturaleza normal o anómala de cada observación nueva en el sistema, tomada del módulo de monitorización y preprocesada, $\hat{\boldsymbol{n}}_{\boldsymbol{r}}$.

El núcleo del sistema de detección consiste en el módulo de *detección de duración y caída anormales*. En este módulo, primeramente, se obtiene F_r de la Ecuación (6.3) para cada recurso observado, r, y se compara con el umbral de caída, F_{th}. Adicionalmente, para aquellos recursos que presenten una caída anormal si su duración T_r es también inferior al umbral de duración D_{th} el sistema lanzará una alarma indicando que el recurso r es potencialmente un recurso *botnet*, en cualquier otro caso el recurso se clasifica como legítimo. Esto es:

$$Clase(r) = \begin{cases} legítimo \;\; si & F_r \leq F_{th} \;\; o \;\; T_r \geq D_{th} \\ botnet \;\; si & F_r > F_{th} \;\; y \;\; T_r < D_{th} \end{cases} \tag{6.7}$$

Nótese en la Figura 6.7, que con el objetivo de incrementar la eficiencia del sistema de detección, previo al módulo de detección de caída anormal del sistema, se realiza un filtrado quedando sólo los recursos populares de acuerdo a la correspondiente etapa de entrenamiento, donde $\theta_P = F_{th}$.

6.6. Evaluación experimental

En esta sección se presentan los resultados de experimentación obtenidos a fin de comprobar el sistema de detección propuesto. Primero, se describen los datos recopilados de la monitorización de la red Mainline y se analizan algunos de los aspectos más destacables de los recursos legítimos monitorizados en base a las siguientes características: dispersión geográfica, popularidad, duración de la compartición y disponibilidad. Después de esto, se evalúan las capacidades del presente sistema de detección. Derivado de ello, se muestran unos patrones reales de evolución en la compartición que presentan, según nuestro modelo, un comportamiento propio de recursos de una *botnet* P2P parásita.

6.6.1. Monitorización de recursos y preprocesado

Siguiendo la metodología explicada en la Sección 6.4, se ha llevado a cabo la monitorización de todos los recursos compartidos en Mainline cuyo identificador de recurso comience por los mismos 8 bits. Concretamente, los 8 bits elegidos han sido `0x8C`. Esto implica que la monitorización ha abarcado 1/256 del total de recursos compartidos a nivel mundial a través de la red Mainline. Este es un tamaño suficientemente representativo que nos permite asumir que el comportamiento de los recursos comprendidos en esta zona de la red de Mainline es comparable al resto de recursos de toda la red.

La monitorización presentada en este capítulo se ha llevado a cabo durante 3 meses, desde el 4 de abril hasta el 4 de julio de 2012. Durante este período de tiempo se han monitorizado un total de 71.135 archivos que han sido compartidos por cientos de millones de direcciones IP diferentes de todos los continentes. Para el muestreo se ha escogido un δ=1 hora. Para cada intervalo δ se almacena la siguiente información por recurso: el número de direcciones IP diferentes que lo comparten, el número de identificadores de nodo diferentes y el número de mensajes de tipo `announce_peer`. Es importante resaltar en este punto que todo este proceso de monitorización es necesario para esta experimentación porque a día de hoy no existen bases de datos conocidas por los autores que recojan la evolución temporal de la compartición de todos los recursos compartidos en una parte representativa de la red Mainline.

Sólo se han tomado en consideración para este estudio los recursos que comenzaron a ser compartidos dentro del periodo de monitorización. Esto se debe a que se pretende recopilar la evolución temporal completa de los recursos, desde el inicio de su compartición hasta el fin. Para filtrar los recursos que comienzan a compartirse en el periodo de monitorización (recursos nuevos), primero se monitoriza durante dos semanas la zona objetivo de Mainline. Después de esta monitorización, todos los recursos que aparezcan y no hayan sido observados en estas dos primeras semanas serán considerados recursos nuevos. Del total de recursos monitorizados, se identifican 34.075 de ellos como nuevos.

Finalmente, se preprocesan los vectores $\hat{\boldsymbol{n}}_r$ siguiendo la Ecuación (6.2) con $W \cdot \delta = 13$ horas. Este tamaño de ventana es suficientemente grande como para disminuir el efecto del elevado *churn* de las redes P2P y suficientemente pequeño para permitir las fluctuaciones propias de la evolución temporal de la compartición de los recursos. Dispuesto el comportamiento de compartición de los recursos, procedemos a su análisis.

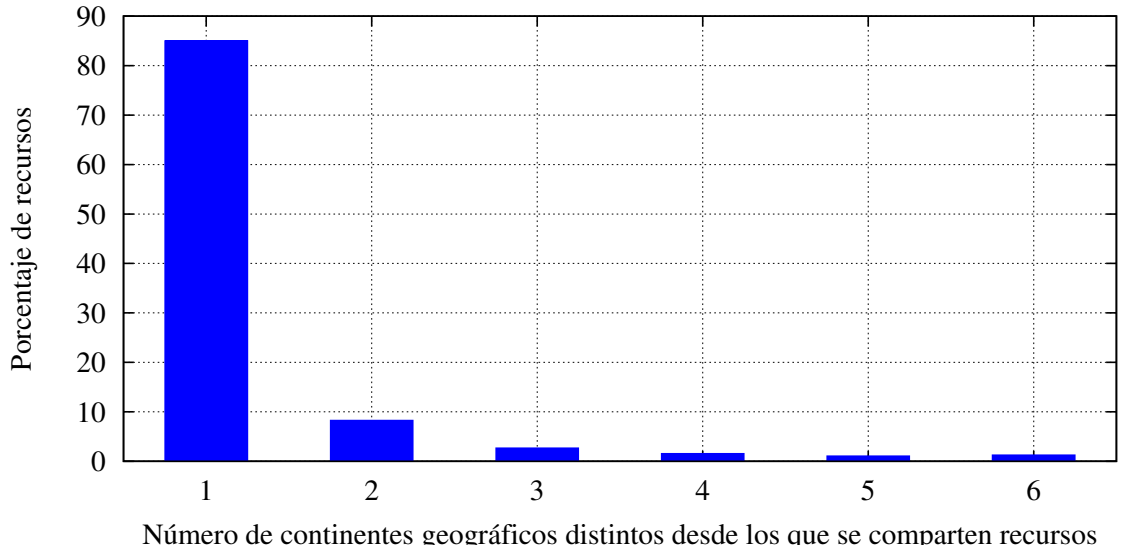
6.6.2. Análisis de los recursos monitorizados en Mainline

A continuación se definen las cuatro características que se analizarán en los recursos monitorizados. Éstas son: (*i*) dispersión geográfica, (*ii*) popularidad, (*iii*) duración de la compartición y (*iv*) disponibilidad. Se considera que estas características pueden ser útiles para realizar un análisis inicial genérico de los datos obtenidos como resultado de la monitorización de la red Mainline.

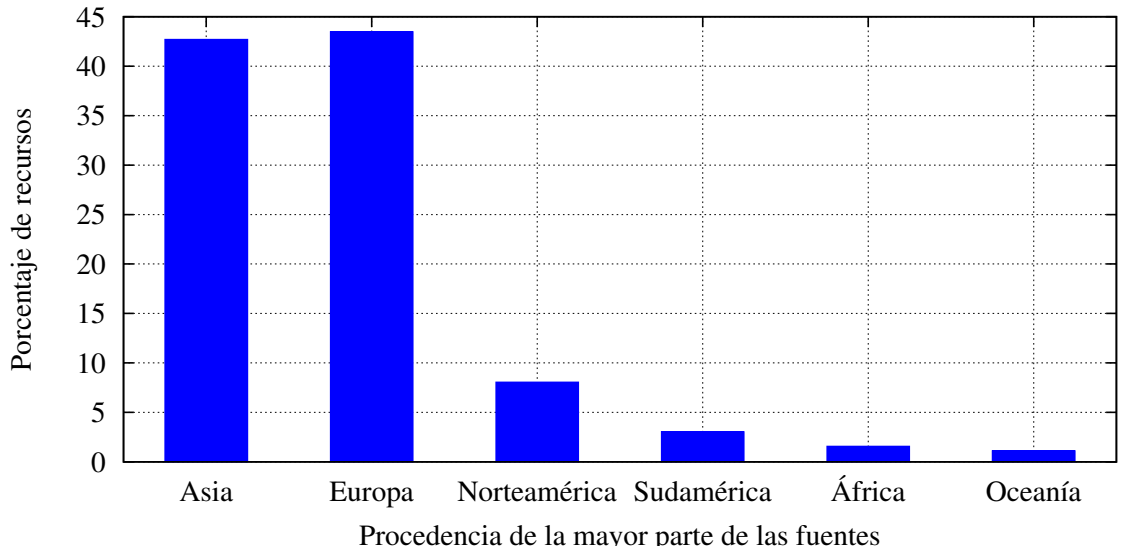
Dispersión geográfica

La dispersión geográfica se extrae como el número de continentes geográficos diferentes desde los que existen nodos compartiendo un recurso concreto. Para esto se han de conseguir en primer lugar las localizaciones de todos los nodos observados en el periodo de monitorización. Para hallar esta correspondencia se ha utilizado el módulo GeoIP de Python [237].

En la Figura 6.9 se presentan los resultados más relevantes relativos a la dispersión geográfica. Podemos ver que el interés de la mayoría de los recursos está centrado en un único continente, con lo que es lógico esperar durante las horas de noche disminuciones importantes del número de usuarios que comparten un recurso (Figura 6.9(a)). En la Figura 6.9(b) se muestra que entre Asia y Europa reúnen más del 80 % de los recursos monitorizados. Se puede concluir que estos continentes son los más activos en el uso de BitTorrent, lo cual es coherente con [2].



(a)



(b)

Figura 6.9: Dispersión geográfica de los recursos monitorizados: (a) número de continentes geográficos diferentes desde los que se comparte un recurso y (b) continentes que presentan la mayoría de las fuentes de los recursos.

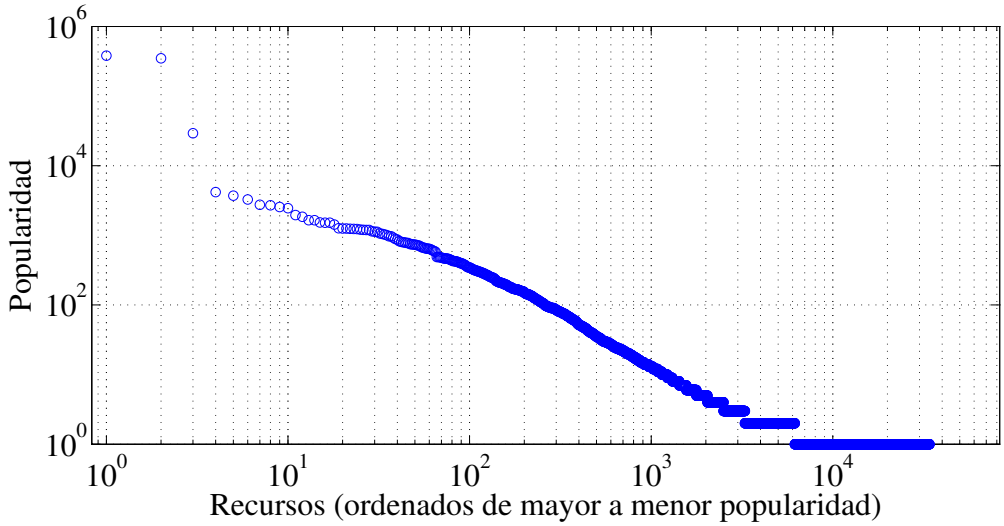


Figura 6.10: Número máximo de ID de nodos diferentes que han compartido los recursos monitorizados.

Popularidad

Definimos la popularidad de un recurso en base al número máximo de nodos diferentes que han compartido dicho recurso durante un δ de una hora. En resumen, los recursos populares serán aquellos que hayan sido compartidos por un mayor número de usuarios. Así la popularidad de un recurso r puede expresarse como, P_r:

$$P_r = \frac{max_k\{n_r(k)\}}{max_{r,k}\{n_r(k)\}};\ \ \forall r \tag{6.8}$$

El rango de P_r es $(0,1]$, siendo $P_r = 1$ el recurso r más popular de los monitorizados y los recursos con P_r cercano a 0 los recursos de menor popularidad.

En la Figura 6.10 se presenta en el eje de abscisas, en escala logarítmica, el orden que ocupan los recursos en función de P_r. Así, en el valor 1 del eje se representa el recurso con $P_r = 1$ y los últimos valores representan a aquellos recursos con menor P_r. En el eje de ordenadas se presenta el número máximo de nodos que han compartido el recurso dado en una hora, $max_k\{n_r(k)\}$.

Como puede verse en esa figura, existen unos pocos recursos con P_r cercano a cero y una enorme cantidad de recursos con P_r cercano a cero. Los diez recursos más populares alcanzan el millón de nodos diferentes compartiéndolos en una hora. Es importante resaltar la elevada cantidad de recursos con $max_k\{n_r(k)\} = 1$; concretamente, en nuestra monitorización encontramos 27.083 recursos en esta situación.

Como se comentó anteriormente, en el sistema de detección de *botnet* parásitas propuesto en este capítulo sólo se utilizan los recursos que poseen una popularidad muy elevada (recursos *top*), que se deriva del umbral de caída anormal. No obstante, con objeto de tomar una referencia del comportamiento de los recursos *top* para el análisis del resto de características y para la posterior generación de datos sintéticos, se asume que son recursos *top* el 1 % de los recursos con un mayor P_r, lo que equivale a un umbral de popularidad, θ_P, de 71 usuarios.

En los 3 meses de monitorización se encuentran 344 recursos *top* (compartidos por más de 71 nodos diferentes en una hora). Si se supone que toda la red Mainline presenta el mismo comportamiento que la zona monitorizada se recopilarían 344 recursos *top* por cada zona en 3 meses, lo que supone 965 recursos *top* nuevos por día en toda la red Mainline. Como es obvio, esta definición de popularidad es bastante relajada dado el elevado número de recursos populares que aparecerían diariamente en base a ella.

Duración de la compartición

La duración de la compartición representa el tiempo durante el cual un recurso ha sido compartido dentro del periodo de monitorización total. Para hallar este valor se restan el último y el primer instante en los que algún nodo de la red envió un mensaje de tipo `announce_peer` relativo al recurso correspondiente. Es importante aclarar que es posible que existan instantes en los que un recurso no esté siendo compartido por nadie y, sin embargo, este tiempo se sume a su duración porque en instantes posteriores algún nodo vuelva a compartirlo.

Como se muestra en la Figura 6.11, más de un 40 % de los recursos monitorizados presenta una duración de la compartición mayor de 20 días. Adicionalmente, se ha comprobado que estos recursos poseen un nivel de popularidad elevado. Este hecho puede verse en la Figura 6.12, en la que se presenta el histograma acumulado de la duración de los 344 recursos etiquetados como populares. De hecho, nótese que el 90 % de ellos presenta una duración mayor de una semana. Como se mencionó en la Sección 6.5.2, el umbral de duración se determina como una proporción de las duraciones de los recursos de la base de datos de entrenamiento. Así, para una proporción β de 3,7 el umbral de duración, D_{th}, sería de 92,5 horas.

Disponibilidad

Para disponer de una medida del tiempo en el que un recurso está siendo realmente compartido, se define la disponibilidad como el tiempo durante el que el recurso estudiado puede ser descargado. Se asume que un recurso puede ser descargado cuando existe al menos un nodo que está compartiendo este recurso. Así, la

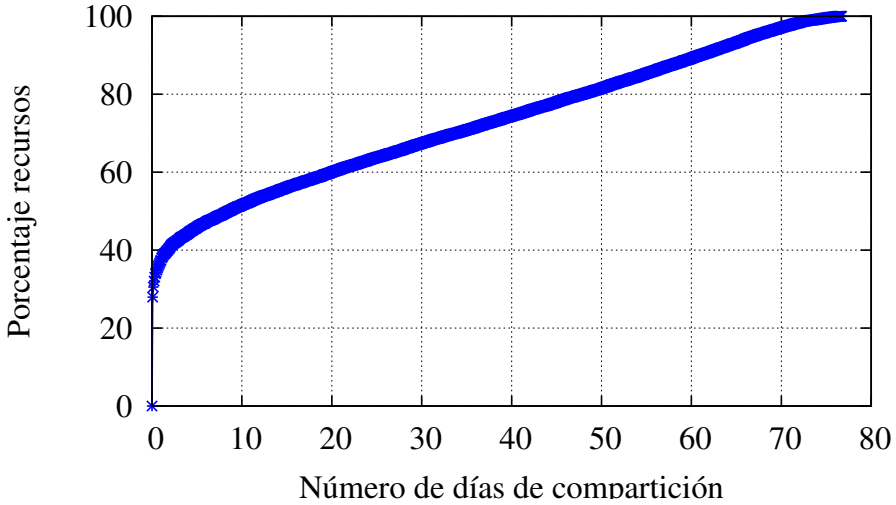


Figura 6.11: Histograma acumulado de la duración de los recursos monitorizados.

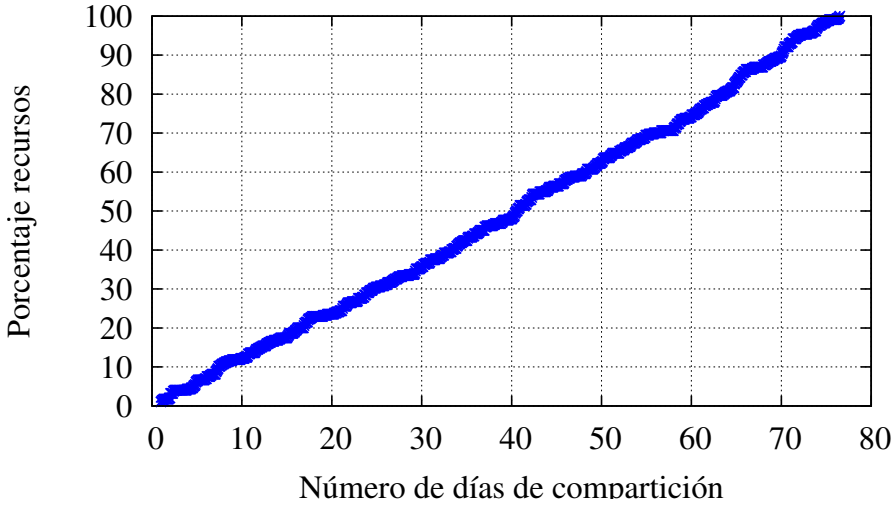


Figura 6.12: Histograma acumulado de la duración de la compartición de los recursos populares monitorizados.

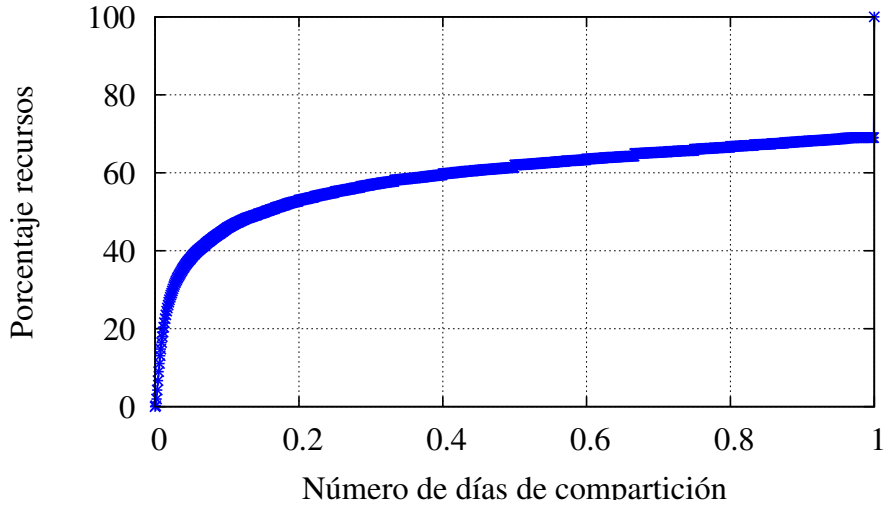


Figura 6.13: Histograma acumulado de la disponibilidad de los recursos monitorizados.

disponibilidad de un recurso, D_r,se calcula como la división del tiempo en el que un recurso dado está siendo compartido, T_r^c, por al menos un usuario entre la duración total del recurso, T_r, quedando como:

$$D_r = \frac{T_r^c}{T_r}$$

Los valores que puede tomar D_r quedan definidos dentro del intervalo (0,1]. Un recurso al ser monitorizado ha sido compartido al menos por un usuario, razón por la que el cero está excluido de los posibles valores de la disponibilidad.

En la Figura 6.13 se muestra el histograma acumulado de la disponibilidad. Alrededor de un 20% de los recursos presentan una disponibilidad cercana a 0. Esto implica que estos recursos poseen una duración elevada pero que han estado disponibles en pocos instantes durante este tiempo. En un estudio sistemático de un porcentaje de estos recursos, se ha comprobado que corresponden en su mayoría con recursos antiguos compartidos por muy pocos nodos, normalmente por uno solamente, de modo que sólo están disponibles el periodo de tiempo en el que los nodos que aún los comparten se conectan a la red.

Cabe destacar en este punto que en un análisis más detallado se descubre que todos los recursos populares monitorizados presentan una disponibilidad cercana a 1 lo que implica que ha sido posible descargarlos durante prácticamente todo su periodo de vida.

Tabla 6.1: Valores para los parámetros de la generación sintética de recursos *botnet*.

	Valores
N	1.000 - 10.000
λ	0,1; 0,3; 0,5
t_{sh}	6 horas - 1 semana
Δ	0,1; 0,2; 0,3

6.6.3. Resultados de detección de recursos *botnets*

De acuerdo con lo expuesto en la Sección 6.5, previamente al proceso de detección es preciso llevar a cabo el entrenamiento de nuestro sistema. Para ello, en primer lugar se identifican aquellos recursos que pueden considerarse como legítimos. Esto se realiza obteniendo información de los recursos publicada en páginas web populares de búsqueda de recursos compartidos por BitTorrent y considerando sólo los que presentan información de buena reputación facilitada por los usuarios. Se han consultado de forma automática 4 páginas web: `http://www.torrentkitty.com/`, `http://torrentproject.com/`, `http://btdigg.org/` y `http://torrentz.eu/`. De los 34.075 recursos, se han seleccionado 14.869 que son considerados como recursos legítimos.

Una vez se ha obtenido la base de datos de entrenamiento (los 14.869 recursos considerados como legítimos), se divide ésta en 4 partes para realizar un proceso de validación cruzada tomando 3 partes para entrenar y la restante para evaluar. Así, se obtienen 4 resultados diferentes de experimentación correspondiendo cada uno a una parte de test diferente. El resultado final de detección del sistema corresponde a la media de estos 4 resultados.

Adicionalmente a las trazas recopiladas de la monitorización de Mainline, se añaden a la partición de test en cada ejecución del procedimiento de validación cruzada 42.000 recursos *botnet* sintéticos siguiendo el modelo presentado en la Sección 6.3. Como se muestra en la Tabla 6.1, se barre un amplio rango de valores para los parámetros de este modelo. Se han escogido valores relativamente bajos de N para comprobar las capacidades de detección del sistema de detección con *botnets* de tamaños reducidos. Se realiza un barrido amplio para las tasas de entrada y salida de los miembros de la *botnet*, λ. Los valores de t_{sh} se han tomado teniendo en consideración que un 1 % de los recursos populares monitorizados en el entrenamiento presentan una duración de al menos 23 horas. Se han escogido valores que van desde las 6 horas hasta 1 semana para probar la robustez del sistema de detección frente a duraciones extensas de los potenciales recursos *bots*. Por último, el intervalo de desaparición de los recursos de una *botnet* varía entre el 0,1 y el 0,3 del tiempo de vida de estos recursos, siendo este tiempo es suficiente para que todos los miembros activos de la *botnet* cumplan la orden de dejar de compartir el recurso.

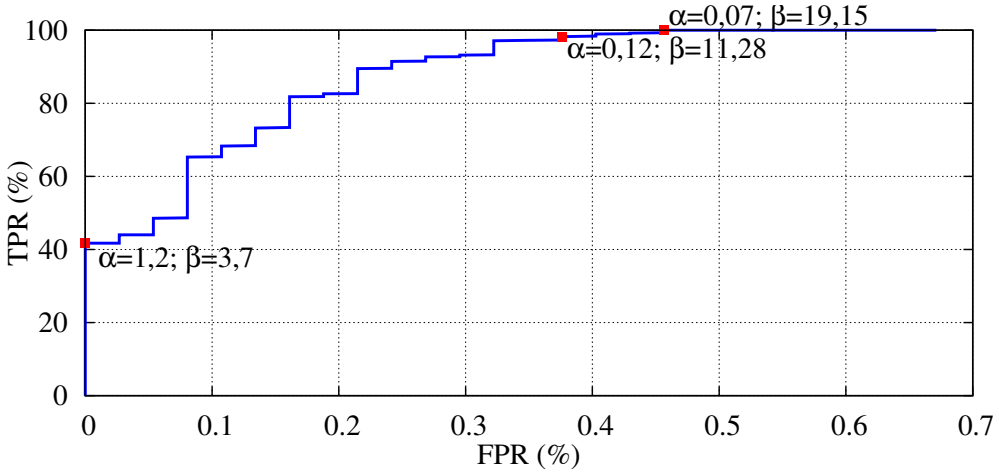


Figura 6.14: ROC del sistema de detección propuesto variando α en Ecuación (6.4) y β en Ecuación (6.6).

Nótese que la popularidad de los recursos *botnet* no es una condición crítica en el sistema de detección, ya que afecta más a la eficiencia del mismo que a su eficacia, esto es, a cuestiones de prestaciones en cuanto a cómputo. Esto es así porque un umbral de popularidad bajo sólo implicará que el filtrado de recursos populares incluye más recursos a procesar pero no afectará a la detección ya que esta condición es ampliamente superada por los recursos *botnet*. La existencia tanto de recursos legítimos como de recursos *botnet* en la base de datos de test permite obtener la curva ROC del sistema de detección.

En base a las Ecuaciones (6.4) y (6.6), la curva ROC ha de obtenerse variando los parámetros α y β de 0,01 a 20. Para obtener una única curva ROC se han escogido parejas de valores *alpha* y *beta* incrementando α y decrementando β dentro del intervalo mencionado anteriormente. Los valores de α y β del intervalo suponen un umbral de popularidad (θ_P) que varía desde 6,26 a 1.252 nodos y un valor de umbral de duración (D_{th}) desde 0,25 horas hasta tres semanas. Se puede ver la ROC resultante en la Figura 6.14, que corresponde a la media de las curvas ROC de los 4 resultados de experimentación relativos al proceso de validación cruzada.

Como se observó, existe un compromiso entre la tasa de falsos positivos (FPR) y la tasa de verdaderos positivos (TPR). De esta forma, para valores de FPR bajos también se reduce TPR, obteniéndose alrededor de un 40 % en TPR para un FPR igual a 0 %. El comportamiento de detección del sistema es suficientemente bueno ya que se alcanza un 98,64 % de tasa de verdaderos positivos para tasas de falsos positivos inferiores a un 0,3 %.

De estos resultados, se puede extraer que un buen punto de operación de este sistema corresponde a un valor de α de 0,12 y β de 11,28 (θ_P igual a 71 nodos y D_{th} igual a 282 horas), lo que implica TPR~99 % y FPR~0,35 %, de acuerdo con la

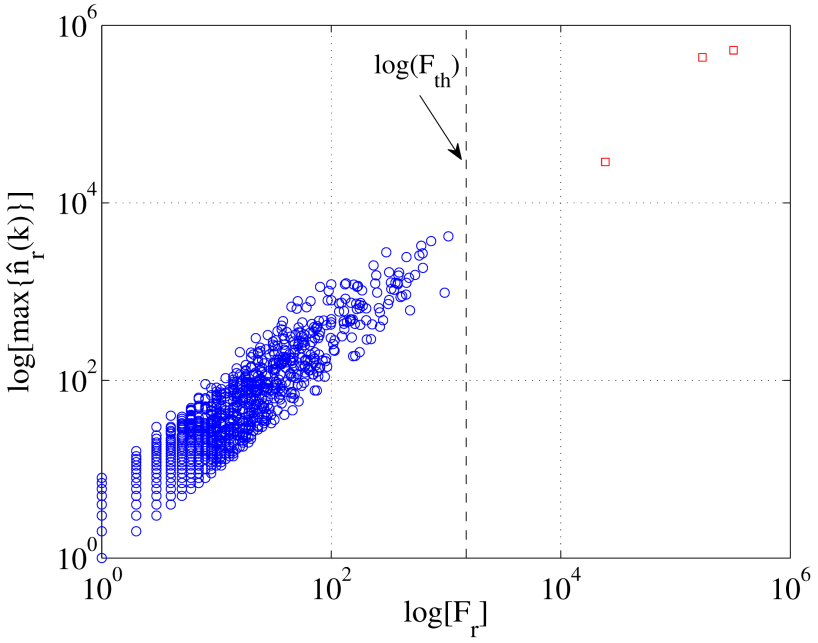


Figura 6.15: Experimentación para la detección de recursos anómalos en la monitorización real de Mainline. Tentativamente aparecen tres recursos anómalos que pueden asociarse a actividad de una *botnet* P2P parásita.

literatura especializada en el campo de sistemas de detección. Sin embargo, este 0,35 % de falsos positivos implicaría unas tres alarmas diarias en la red Mainline y, dado que se asume que una *botnet* necesita enviar órdenes constantemente a sus *bots*, la tasa de verdaderos positivos no es una restricción crítica. Esto es así porque, como se comentó con anterioridad, los comandos del *botmaster* son muy frecuentes y un *bot* debe estar siempre actualizado, lo que implica que existen múltiples oportunidades para detectar recursos *botnet*. Por ello, aquí se considera como punto de operación más adecuado a este caso particular el correspondiente a un valor de α igual a 1,2 y β igual a 3,7 (θ_P igual a 756 nodos y D_{th} igual a 92,5 horas), lo que implica una tasa de falsos positivos igual a cero y una tasa de verdaderos positivos del 40 %.

6.6.4. Descubrimiento de patrones de actividad de *botnets* P2P parásitas

Una vez que se ha mostrado el comportamiento de la aproximación de detección aquí propuesta, se realiza un experimento adicional para comprobar el sistema cuando se analizan nuevos recursos con el detector convenientemente ajustado tras los procesos de entrenamiento y detección antes descritos.

Para esto se utilizan los restantes 19.206 recursos que se obtuvieron en el proceso de monitorización y que, al no ser clasificados como legítimos, no fueron utilizados

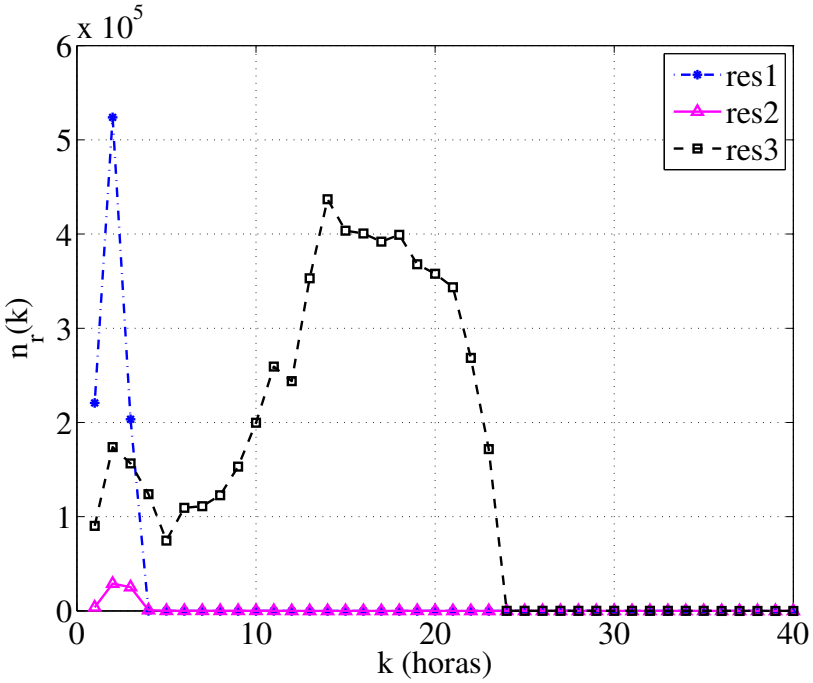


Figura 6.16: Evolución temporal de los tres recursos detectados como anómalos (δ = 1 hora).

ni en el entrenamiento ni en el test. Estos recursos podrían corresponder tanto a recursos *botnet* como a recursos legítimos, ya que lo único que sabemos de ellos es que no han sido explícitamente votados por los usuarios como legítimos en las páginas web consultadas. El punto de operación escogido para esta experimentación es el correspondiente a un valor de $\alpha = 1{,}2$ y $\beta = 3{,}7$, lo que implica una tasa de falsos positivos teórica del 0 %. Este ha sido el punto de operación escogido según se ha indicado anteriormente para ser muy restrictivos en la generación de alarmas y, así, aumentar la certeza acerca de la naturaleza maliciosa de los eventos detectados.

En la Figura 6.15 se representa, para todos los recursos, el máximo valor de popularidad alcanzado, $max\{n_r(k)\}$ (con el eje y en escala logarítmica), y la caída obtenida de la Ecuación (6.3) (con el eje x también en escala logarítmica). Aquí, el umbral de caída F_{th} también se representa en valor logarítmico con una línea discontinua. Véase que el sistema de detección sólo genera alarmas para tres de los recursos monitorizados, los cuales claramente se comportan como *outliers*, lejos de la distribución común de los datos.

En la Figura 6.16 se muestra $\boldsymbol{n_r}$ para estos tres recursos. Aquí se puede comprobar que todos ellos son compartidos por un enorme número de usuarios durante un periodo de tiempo muy reducido. Concretamente, *res1* alcanza un máximo de 523.883 usuarios y la fase de compartición dura unas 5 horas. Algo similar sucede con el

recurso *res2*, con un número menor de usuarios pero aún extremadamente alto (27,083). En cuanto a *res3*, la duración de la fase de compartición es un poco superior aunque sigue siendo muy reducida, alrededor de 24 horas, y el número máximo de nodos que lo comparte llega a 436.963.

El comportamiento de la evolución de estos recursos es altamente sospechoso de pertenecer a la compartición de recursos de una *botnet*, ya que siguen claramente el modelo presentado en la Sección 6.3. Desafortunadamente, no se ha podido comprobar la veracidad de esta asunción debido a que la corta duración de vida de estos recursos impidió su descarga y posterior análisis por medio de técnicas de ingeniería inversa.

6.7. Conclusiones

Este capítulo presenta una aproximación novedosa para determinar la ocurrencia de eventos de *botnets* P2P parásitas. La principal novedad del presente sistema de detección es que se basa en la monitorización de la evolución temporal de los recursos compartidos en las redes P2P en lugar de en la monitorización del tráfico de red en sí. Nosotros aseguramos que los recursos legítimos compartidos a través de redes de compartición de recursos P2P pueden presentan una evolución de su compartición que difiere sustancialmente de la evolución de los recursos *botnets*, relacionados con las comunicaciones de éstas.

Para probar la validez de nuestra hipótesis, se ha desarrollado una metodología para recuperar información de los patrones de los recursos compartidos en una red P2P. Ésta metodología se ha comprobado en la red Mainline, probando su viabilidad. El sistema de monitorización ha permitido la definición de modelos de patrones de compartición tanto para recursos legítimos como para recursos *botnets*. A partir de estos se ha desarrollado un sistema de detección basado en la caída abrupta del número de nodos que comparten un recurso dado.

Como prueba de concepto, se ha recopilado la evolución temporal de la compartición de los recursos de una zona de la red Mainline durante 3 meses. Primeramente, se analizan los recursos monitorizados en base a 4 características, a saber: dispersión geográfica, popularidad, duración de la compartición y disponibilidad. Posteriormente, se muestra que el sistema propuesto exhibe unas grandes capacidades, tanto en términos de falsos positivos (menores de un 0,4%), como en términos de tasa de aciertos (mayor del 99%). En una primera utilización del sistema de detección propuesto, tres de los recursos monitorizados han sido detectados como anómalos y presentan un compartimiento de compartición propio de recursos *botnets*. Finalmente, la simplicidad de la solución propuesta la hace adecuada para su implementación en sistemas reales.

Capítulo 7

Conclusiones y trabajo futuro

En el presente capítulo se resumen las principales conclusiones extraídas tras la realización del trabajo completo de tesis aquí expuesto. Éstas se han ido detallando más concretamente en cada uno de los capítulos previos, presentándose en éste de forma unificada y sintética. Adicionalmente, se describen brevemente los principales trabajos futuros a abordar en esta línea de investigación.

7.1. Conclusiones

El volumen de tráfico asociado a las redes P2P, y más concretamente a las aplicaciones basadas en éstas, representa hoy en día un porcentaje muy significativo de todo el tráfico de la red. Sin embargo, la seguridad en estas redes es aún hoy un problema sin solución. Un primer paso en la provisión de servicios de seguridad en estas redes es la identificación del tráfico que viaja por la red, para que así, por ejemplo, los ISP puedan establecer unas políticas de gestión del tráfico u otras en función del tipo de tráfico concreto que viaje por sus redes. En relación a ello:

- Se ha comenzado presentando una breve revisión bibliográfica de la investigación relativa a la clasificación de tráfico. Se ha puesto un énfasis especial en los trabajos relativos a la clasificación de tráfico P2P.
- Se han presentado dos sistemas de clasificación de tráfico: uno específico para el protocolo eDonkey y otro susceptible de adaptación para la clasificación de cualquier protocolo de red.

- La propuesta de detección del tráfico eDonkey se basa en el comportamiento particular de este protocolo. Para este fin se han propuesto dos algoritmos de detección basados en heurísticas: el primero tiene como objeto la detección de flujos eDonkey (WCFD) y el segundo la detección de nodos generadores de tráfico eDonkey (URND).

- El algoritmo de detección de flujos, WCFD, se basa en la suposición de que la información enviada desde el cliente que inicia la conexión al que la recibe es sustancialmente mayor que en el sentido inverso. También se ha podido concluir que este algoritmo de detección sólo es válido en la detección de flujos de compartición de archivos para nodos eDonkey con un identificador ID alto.

- El algoritmo de detección de nodos, URND, se basa en que los nodos generadores de tráfico eDonkey presentan una tasa de subida constante y se comunican con múltiples usuarios a la vez.

- El modelo susceptible de adaptación para la detección de otros protocolos sigue una metodología basada en el uso de modelos ocultos de Markov (HMM) para clasificar ciertos servicios. Así, se contribuye con un HMM genérico a nivel de flujo y cuyas observaciones son los paquetes del flujo. Este HMM genérico se concreta aquí para la clasificación de tráfico del protocolo eDonkey como caso de estudio.

- Se ha realizado una experimentación con cuatro conjuntos de tráfico, todos ellos de comunicaciones reales y dos de ellos capturados en condiciones controladas. Se puede concluir que tanto los algoritmos de detección para el tráfico eDonkey como el sistema de clasificación basado en HMM presentan una buena tasa de detección y un reducido número de falsos positivos.

- Ambos sistemas de clasificación son capaces de identificar tráfico sin inspeccionar el contenido de los paquetes y, por tanto, respetando las leyes de protección de la privacidad. Además, se ha comprobado que son capaces de detectar comunicaciones ofuscadas.

Tras la identificación del tráfico P2P, se ha argumentado que es posible aplicar tanto políticas de seguridad como sistemas de detección de anomalías sobre el tráfico identificado para proteger las redes frente a ataques a la seguridad de las mismas. Uno de los problemas actuales identificados como relevantes en cuanto a la seguridad de las redes P2P son las *botnets* o redes de *zombies*. Las *botnets* son un tema de creciente importancia y así se ve reflejado en el interés de la comunidad investigadora en este campo. En esta línea, las principales conclusiones del presente trabajo se resumen a continuación:

- Se ha propuesto un ciclo de vida genérico de las *botnets* con una idea fundamental: todas las etapas que lo forman han de ser ejecutadas satisfactoriamente para alcanzar el objetivo final de la *botnet*, esto es, que el ataque para el que ha sido concebida tenga éxito. En base a este ciclo de vida se propone la taxonomía anteriormente comentada. Se ha discutido que la interrupción de la ejecución de al menos una de las etapas del ciclo de vida conseguirá frustrar la actividad de la *botnet* completa o, cuando menos, reducir notablemente sus efectos.

- Se ha propuesto una nueva taxonomía basada en el ciclo de vida anterior. Esta taxonomía tiene como objetivo fundamental agrupar la investigación en *botnets* y así aportar una visión global de la problemática de las mismas y organizar la enorme cantidad de trabajos en este tema.

- Siguiendo la taxonomía anterior se han revisado los trabajos de investigación más relevantes del campo de las *botnets* aportando una visión general de las contribuciones más importantes del campo.

- Se ha contribuido con una visión general de la problemática de las *botnets* que ha permitido revelar las deficiencias que aún han de ser tenidas en cuenta por parte de la comunidad investigadora en los próximos años. Se han indicado también los retos de investigación a abordar, junto a algunas líneas que puedan servir de ayuda para afrontarlos.

- Seguidamente, se ha planteado como línea de investigación detectar nuevas amenazas en el campo de las *botnets* de forma temprana; concretamente, identificar un nuevo tipo de *botnets* denominadas *botnet* P2P parásitas.

- Se ha presentado una aproximación novedosa para determinar la ocurrencia de eventos de *botnets* P2P parásitas. La principal novedad del sistema de detección propuesto es que está basado en la monitorización de la evolución temporal de los recursos compartidos en las redes P2P en lugar de en la monitorización del tráfico de red en sí mismo. Los recursos legítimos compartidos a través de redes de compartición de recursos P2P presentan una evolución de su compartición que difiere sustancialmente de la evolución de los recursos *botnets*, relacionados con las comunicaciones de éstas.

- Para probar la validez de nuestra hipótesis, se ha desarrollado una metodología para recuperar información de los patrones de los recursos compartidos en una red P2P. Esta metodología se ha probado en la red Mainline, comprobando su viabilidad. El sistema de monitorización propuesto ha permitido la definición de modelos de patrones de compartición tanto para recursos legítimos como para recursos *botnet*. Para éstos, se ha desarrollado un sistema de detección basado en la caída abrupta del número de nodos que comparten un recurso dado.

- La experimentación realizada a partir de todo lo anterior demuestra que el sistema propuesto exhibe unas grandes capacidades de detección, tanto en términos de falsos positivos (menores de un 0,4%) como en términos de tasa de aciertos (mayor del 99%). Adicionalmente, la simplicidad de la solución propuesta la hace adecuada para su implementación en sistemas reales.

7.2. Líneas de trabajo futuro

Las principales líneas de trabajo futuro que se extraen del presente documento se enumeran a continuación:

1. Explorar la posibilidad de detectar otros protocolos de compartición de recursos mediante P2P (como BitTorrent) utilizando el algoritmo de detección de nodos URND. Es asumible que esta detección pueda extrapolarse, dado que los protocolos de compartición de recursos saturan la tasa de subida del usuario y esto implica la necesidad de una limitación en la misma.

2. Ampliar la detección basada en la limitación de la tasa de subida de las aplicaciones de compartición de recursos mediante P2P, con el desarrollo de un nuevo algoritmo basado en la tasa de subida constante que estas aplicaciones experimentan con cada uno de los usuarios con los que comparten recursos. No sólo la tasa de subida global es constante, sino que además esta tasa de subida se reparte de forma equitativa entre los distintos usuarios con los que se comparte algún recurso. Esta característica se cree que puede ser una característica distintiva de este tipo de aplicaciones.

3. Extender la experimentación del sistema de clasificación de tráfico basado en HMM para probar su capacidad de detección con respecto a otros protocolos distintos del protocolo eDonkey.

4. Aunar en un único sistema de detección los algoritmos de detección URND y WCFD junto con la clasificación basada en modelado de Markov. De esta forma será posible, en primer lugar, detectar nodos generadores de tráfico eDonkey y, en segundo lugar, identificar los flujos concretos de estos nodos que corresponden a este protocolo.

5. Utilizar el sistema de clasificación de tráfico basado en HMM con la información monitorizada por el módulo de *sniffing* desarrollado para Mainline. De la misma forma que la evolución temporal de la compartición de los recursos de tipo *botnet* presenta unas características diferenciadoras, creemos que el comportamiento de compartición de los recursos de distintos tipos (audio, vídeo, programas, etc.) presentará también unas características concretas que

podrán ser recogidas con un modelado de Markov. Esto sería de gran utilidad en múltiples campos como, por ejemplo, la detección de la compartición de recursos protegidos con *copyright*.

6. Automatizar completamente el sistema de detección de *botnet* P2P parásitas, incluyendo la descarga y posterior análisis de los recursos que generen alarmas en el sistema para así poder comprobar si los recursos detectados forman parte o no de las comunicaciones de una *botnet* P2P parásita.

7. Aumentar la eficiencia del sistema de monitorización para poder monitorizar la red completa de Mainline y así detectar la actividad de *botnets* parásitas con mayor eficacia.

8. Desarrollar un sistema de detección de *botnets* parásitas cuyas interacciones derivadas de sus actualizaciones se realicen entre grupos reducidos de nodos, a fin de evitar la detección por parte del presente sistema. El diseño del sistema de detección de este tipo de *botnets* debe basarse en el aumento del número de archivos nuevos generados y compartidos por un mismo subconjunto de nodos de la red.

Bibliografía

[1] Apu Kapadia. Halo: High-assurance locate for distributed hash tables. In *In The 15th Annual Network and Distributed System Security Symposium (NDSS), To Appear*, 2008.

[2] Hendrik Schulze and Klaus Mochalski. Internet Study 2008/2009. Technical report, 2009.

[3] Karim El Defrawy, Minas Gjoka, and Athina Markopoulou. BotTorrent: Misusing BitTorrent to Launch DDoS Attacks. In *Proceedings of the 3rd USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet*, SRUTI'07, 2007.

[4] Norton Symantec. Norton Study Calculates Cost of Global Cybercrime: $114 Billion Annually. `http://www.symantec.com/about/news/release/article.jsp?prid=20110907_02`, 2011. [Online; Last accessed January-7th-2014].

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, 1998. RFC 2475.

[6] M. Feily, A. Shahrestani, and S. Ramadass. A Survey of Botnet and Botnet Detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 268–273, 2009.

[7] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 41–52, New York, NY, USA, 2006. ACM.

[8] Tim Weber. Criminals 'may overwhelm the web'. Technical report, BBC News, January 2007.

[9] Norton Symantec. Symantec Global Internet Security Threat Report trends for July - December 07. `http://www.symantec.com/business/theme.jsp?themeid=threatreport`, April 2008. [Online; Last accessed January-7th-2014].

[10] Mcaffe. McAfee Threats Report: First Quarter 2009. `http://resources.mcafee.com/content/AvertReportQ109`. [Online; Last accessed January-7th-2014].

[11] Scopus. Bibliographic database containing academic articles. `http://www.scopus.com`. [Online; Last accessed January-7th-2014].

[12] Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, and Akira Kato. The Impact and Implications of the Growth in Residential User-to-user Traffic. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 207–218, 2006.

[13] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan. Characterizing User Behavior and Network Performance in a Public Wireless LAN. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '02, pages 195–205, 2002.

[14] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 35 of *SIGCOMM '05*, pages 229–240, New York, NY, USA, 2005. ACM.

[15] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, volume 33 of *SIGMETRICS '05*, pages 50–60, New York, NY, USA, June 2005. ACM.

[16] OpenDPI. `http://www.opendpi.org/`. [Online; Last accessed January-7th-2014].

[17] A. Madhukar and C. Williamson. A Longitudinal Study of P2P Traffic Classification. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*, pages 179–188, 2006.

[18] Hongwei Chen, Xin Zhou, Fangping You, and Chunzhi Wang. Study of Double-Characteristics-Based SVM Method for P2P Traffic Identification. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 1, pages 202–205, April 2010.

[19] Jae Yoon Chung, Byungchul Park, Y.J. Won, J. Strassner, and J.W. Hong. An effective similarity metric for application traffic classification. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 286–292, April 2010.

[20] C. Rottondi and G. Verticale. Using packet interarrival times for Internet traffic classification. In *Communications (LATINCOM), 2011 IEEE Latin-American Conference on*, pages 1–6, Oct. 2011.

[21] V.P. de A Ribeiro, R.H. Filho, and J.E.B. Maia. Online traffic classification based on sub-flows. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 415–421, May 2011.

[22] Xu Tian, Qiong Sun, Xiaohong Huang, and Yan Ma. A Dynamic Online Traffic Classification Methodology Based on Data Stream Mining. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 1, pages 298–302, 2009.

[23] Hongwei Chen, Zhengbing Hu, Zhiwei Ye, and Wei Liu. Research of P2P Traffic Identification Based on Neural Network. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–4, Jan. 2009.

[24] F. Dehghani, N. Movahhedinia, M.R. Khayyambashi, and S. Kianian. Real-Time Traffic Classification Based on Statistical and Payload Content Features. In *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, pages 1–4, May 2010.

[25] Arthur Callado, Judith Kelner, Djamel Sadok, Carlos Alberto Kamienski, and Stênio Fernandes. Better network traffic identification through the independent combination of techniques. *J. Netw. Comput. Appl.*, 33:433–446, July 2010.

[26] Arthur Callado, Carlos Kamienski, Geza Szabo, Balazs Gero, Judith Kelner, Stenio Fernandes, and Djamel Sadok. A Survey on Internet Traffic Identification. *IEEE Communications Surveys & Tutorials*, 11(3):37–52, August 2009.

[27] Ram Keralapura, Antonio Nucci, and Chen-Nee Chuah. A novel self-learning architecture for p2p traffic classification in high speed networks. *Comput. Netw.*, 54:1055–1068, May 2010.

[28] IANA. Port Numbers. `http://www.iana.org/assignments/port-numbers`. [Online; accessed January-7th-2013].

[29] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate scalable in-network identification of P2P traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 512–521, New York, NY, USA, 2004. ACM.

[30] A. Dainotti, A. Pescape, and K.C. Claffy. Issues and future directions in traffic classification. *Network, IEEE*, 26(1):35–40, February 2012.

[31] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and K.C. Claffy. Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 121–134, New York, NY, USA, 2004. ACM.

[32] A. Spognardi, A. Lucarelli, and R. Di Pietro. A methodology for P2P file-sharing traffic detection . In *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on*, pages 52–61, 2005.

[33] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 135–148, New York, NY, USA, 2004. ACM.

[34] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning . In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 250–257, 2005.

[35] J. Erman, A. Mahanti, and M. Arlitt. QRP05-4: Internet Traffic Identification using Machine Learning . In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–6, 2006.

[36] Nigel Williams, Sebastian Zander, and Grenville Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, October 2006.

[37] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, pages 6:1–6:12, New York, NY, USA, 2006. ACM.

[38] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic Classification on the Fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, April 2006.

[39] Laurent Bernaille and Renata Teixeira. Early Recognition of Encrypted Applications. In *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, PAM'07, pages 165–175, 2007.

[40] G. Szabo, Istvan Szabo, and Daniel Orincsay. Accurate Traffic Classification . In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, 2007.

[41] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype traffic: when randomness plays with you. In *Proceedings of*

the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '07, 2007.

[42] Kyoungwon Suh, D.R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and Detecting Skype-Relayed Traffic . In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.

[43] Hamza Dahmouni, Sandrine Vaton, and David Rossé. A markovian signature-based approach to IP traffic classification. In *Proceedings of the 3rd annual ACM workshop on Mining network data*, MineNet '07, pages 29–34, New York, NY, USA, 2007. ACM.

[44] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *J. Mach. Learn. Res.*, 7:2745–2769, December 2006.

[45] A. Dainotti, W. de Donato, A. Pescape, and P. Salvo Rossi. Classification of Network Traffic via Packet-Level Hidden Markov Models. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE, November 2008.

[46] K. Thompson, G.J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *Network, IEEE*, 11(6):10–23, Nov/Dec 1997.

[47] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling Internet backbone traffic: behavior models and applications. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 169–180, New York, NY, USA, 2005. ACM.

[48] Ke Xu, Ming Zhang, Mingjiang Ye, Dah M. Chiu, and Jianping Wu. Identify P2P traffic by inspecting data transfer behavior. *Computer Communications*, 33(10):1141–1150, June 2010.

[49] Augustin Soule, Kavé Salamatian, and Nina Taft. Combining Filtering and Statistical Methods for Anomaly Detection. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, 2005.

[50] Chin-Tser Huang, S. Thareja, and Yong-June Shin. Wavelet-based Real Time Detection of Network Traffic Anomalies . In *Securecomm and Workshops, 2006*, 2006.

[51] Antonio Magnaghi, Takeo Hamada, and Tsuneo Katsuyama. A Wavelet-based Framework for Proactive Detection of Network Misconfigurations. In *Proceedings of the ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice Meet Malfunctioning Reality*, NetT '04, 2004.

[52] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining Anomalies Using Traffic Feature Distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4), August 2005.

[53] A. Feldmann. A possibility for ISP and P2P collaboration. In *Broadband Communications, Networks and Systems, 2008. BROADNETS 2008. 5th International Conference on*, page 239, 2008.

[54] J. Ramirez, J.C. Segura, C. Benitez, A. de la Torre, and A.J. Rubio. A new Kullback-Leibler VAD for speech recognition in noise. *Signal Processing Letters, IEEE*, 11(2):266–269, 2004.

[55] J. Astola, P. Haavisto, and Y. Neuvo. Vector median filters. *Proceedings of the IEEE*, 78(4):678–689, April 1990.

[56] A.A. Markov and N.M. Nagorny. *The theory of algorithms*. Mathematics and its applications (Kluwer Academic Publishers): Soviet series. Kluwer Academic, 1988.

[57] Przemyslaw Dymarski. *Hidden Markov Models Theory and Applications*. InTech, 2011.

[58] G.A. Fink. *Markov models for pattern recognition: from theory to applications*. Springer, 2008.

[59] A. Feldmann. Characteristics of TCP Connection Arrivals. Technical memorandum, AT&T Labs Research, 1998.

[60] R. A. Johnson and D. W. Wichern, editors. *Applied multivariate statistical analysis*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[61] Yoram Kulbak and Danny Bickson. The eMule Protocol Specification. Technical report, 2005.

[62] aMule. `http://www.amule.org/`. [Online; Last accessed January-7th-2014].

[63] Tyler Moore, Richard Clayton, and Ross Anderson. The Economics of Online Crime. *The Journal of Economic Perspectives*, 23(3):3–20, 2009.

[64] Clay Wilson. Botnets Cybercrime and Cyberterrorism: Vulnerabilities and Policy Issues for Congress. `www.fas.org/sgp/crs/terror/RL32114.pdf`, November 2007. [Online; Last accessed January-7th-2014].

[65] D. Dagon, Guofei Gu, C.P. Lee, and Wenke Lee. A Taxonomy of Botnet Structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339, 2007.

[66] H.R. Zeidanloo, M.J.Z. Shooshtari, P.V. Amoli, M. Safari, and M. Zamani. A taxonomy of Botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, pages 158–162, 2010.

[67] Chao Li, Wei Jiang, and Xin Zou. Botnet: Survey and Case Study. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*, pages 1184–1187, 2009.

[68] Zhaosheng Zhu, Guohan Lu, Yan Chen, Z.J. Fu, P. Roberts, and Keesook Han. Botnet Research Survey. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 967–972, 2008.

[69] M. Bailey, E. Cooke, F. Jahanian, Yunjing Xu, and M. Karir. A Survey of Botnet Technology and Defenses. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 299–304, 2009.

[70] Fortinet. Fortinet August Threat Landscape Report Shows Return of Ransomware and Rise of Do-It-Yourself Botnets. `http://investor.fortinet.com/releasedetail.cfm?releaseid=504094`, September 2010. [Online; Last accessed January-7th-2014].

[71] Chris Boyd. The DIY Twitter Botnet Creator. `http://www.gfi.com/blog/the-diy-twitter-botnet-creator/`, May 2010. [Online; Last accessed July-23th-2013].

[72] Dancho Danchev. DIY botnet kit spotted in the wild. `http://www.zdnet.com/blog/security/diy-botnet-kit-spotted-in-the-wild/9440`, September 2010. [Online; Last accessed January-7th-2014].

[73] Wendy McElroy. In Child Porn Case Technology Entraps the Innocent. `http://www.foxnews.com/story/0,2933,244009,00.html`, 2007. [Online; Last accessed January-7th-2014].

[74] Jing Liu, Yang Xiao, Kaveh Ghaboosi, Hongmei Deng, and Jingyuan Zhang. Botnet: classification attacks detection tracing and preventive measures. *EURASIP Journal on Wireless Communications and Networking*, 2009.

[75] J. Govil and G. Jivika. Criminology of BotNets and their detection and defense methods. In *Electro/Information Technology, 2007 IEEE International Conference on*, pages 215–220, May 2007.

[76] Edward Balas. *Know your Enemy: Learning about Security Threats*. Addison Wesley Publishing, 2nd edition edition, 2004.

[77] Norton Symantec. Symantec Global Internet Security Threat Report trends of 2009. http://www.symantec.com/business/theme.jsp?themeid=threatreport, April 2010. [Online; Last accessed January-7th-2014].

[78] R. Pointer. Home page of Eggdrop botnet. http://www.eggheads.org/, 1993. [Online; Last accessed January-7th-2014].

[79] Joe Stewart. Bobax Trojan Analysis. http://www.secureworks.com/research/threats/bobax/, May 2004. [Online; Last accessed January-7th-2014].

[80] F. Leder and T. Werner. Know Your Enemy: Containing Conficker. http://www.honeynet.org/files/KYE-Conficker.pdf, April 2009. [Online; Last accessed January-7th-2014].

[81] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–8, Berkeley, CA, USA, 2007. USENIX Association.

[82] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. In *USENIX; login*, volume 32, pages 46–63, December 2007.

[83] Joe Stewart. Phatbot trojan Analysis. http://www.secureworks.com/research/threats/phatbot/, March 2004. [Online; Last accessed July-23th-2013].

[84] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[85] G. Sinclair, C. Nunnery, and B.B.-H. Kang. The waledac protocol: The how and why. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 69–77, 2009.

[86] Ping Wang, S. Sparks, and C.C. Zou. An Advanced Hybrid Peer-to-Peer Botnet. *Dependable and Secure Computing, IEEE Transactions on*, 7(2):113–127, 2010.

[87] Eugene Rodionov and Aleksandr Matrosov. The Evolution of TDL: Conquering x64. http://www.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf, June 2011. [Online; Last accessed January-7th-2014].

[88] abuse.ch. ZeuS Gets More Sophisticated Using P2P Techniques. http://www.abuse.ch/?p=3499, October 2011. [Online; Last accessed January-7th-2014].

[89] NVD. Vulnerabilities in the last three years. `http://nvd.nist.gov/`, 2010. [Online; Last accessed January-7th-2014].

[90] Eric Chien. W32.Stuxnet Dossier. `http://www.symantec.com/connect/es/blogs/w32stuxnet-dossier`, September 2010. [Online; Last accessed January-7th-2014].

[91] Paul Barford and Vinod Yegneswaran. An Inside Look at Botnets. In *ARO-DHS Special Workshop on Malware Detection*, volume 27 of *Advances in Information Security*, chapter 8, pages 171–191. Springer US, Boston, MA, 2007.

[92] Niels Provos, Moheeb Abu Rajab, and Panayiotis Mavrommatis. Cybercrime 2.0: when the cloud turns dark. *Commun. ACM*, 52:42–47, April 2009.

[93] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to Us. In *Proceedings of the 17th conference on Security symposium*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[94] Alan Solomon and Gadi Evron. The world of botnets. *Virus Bulletin*, pages 10–12, September 2006.

[95] M.R. Faghani and H. Saidi. Malware propagation in Online Social Networks. In *4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 8–14, 2009.

[96] Kim Zetter. Trick or Tweet? Malware Abundant in Twitter URLs. Technical report, Kaspersky, October 2009.

[97] Joe Stewart. Sinit P2P trojan analysis. `http://www.securiteam.com/securityreviews/6J0022A95E.html`, December 2009. [Online; Last accessed January-7th-2014].

[98] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[99] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, volume 2, pages 49–54. USENIX Association, 2006.

[100] Turkojan. DIY kit of Turkojan 4. `http://www.turkojan.com/eng/`, 2012. [Online; Last accessed January-7th-2014].

[101] FBI. Another Pleads Guilty in Botnet Hacking Conspiracy. `http://dallas.fbi.gov/dojpressrel/pressrel10/dl061010.htm`, June 2010. [Online; Last accessed January-7th-2014].

[102] Yury Namestnikov. The economics of Botnets. `http://www.securelist.com/en/analysis?pubid=204792068`, July 2009. [Online; Last accessed January-7th-2014].

[103] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[104] Gordon V. Cormack. Email Spam Filtering: A Systematic Review. *Found. Trends Inf. Retr.*, 1:335–455, April 2008.

[105] Alta van der Merwe, Marianne Loock, and Marek Dabrowski. Characteristics and responsibilities involved in a Phishing attack. In *Proceedings of the 4th international symposium on Information and communication technologies*, WISICT '05, pages 249–254. Trinity College Dublin, 2005.

[106] Joe Stewart. ZeuS Banking Trojan Report. `http://www.secureworks.com/research/threats/zeus`, March 2010. [Online; Last accessed January-7th-2014].

[107] Kenneth C. Wilbur and Yi Zhu. Click Fraud. *Marketing Science*, 28:293–308, March 2009.

[108] Neil Daswani and Michael Stoppelman. The anatomy of Clickbot.A. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, 2007.

[109] Joe Stewart. SpamThru Trojan Analysis. `http://www.secureworks.com/research/threats/spamthru/`, October 2006. [Online; Last accessed January-7th-2014].

[110] Igor V. Popov, Saumya K. Debray, and Gregory R. Andrews. Binary obfuscation using signals. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 275–290. USENIX Association, 2007.

[111] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm. `www.cyber-ta.org/pubs/StormWorm/`, 2007. [Online; Last accessed January-7th-2014].

[112] APEC. Guide on Policy and Technical Approaches against Botnet. Technical report, Telecommunications and Information Working Group Asia-Pacific Economic Cooperation (APEC), 2008.

[113] Felix Freiling, Thorsten Holz, and Georg Wicherski. Botnet Tracking: Exploring a root-cause methodology to prevent denial-of-service attacks. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS '05)*, pages 319–335, September 2005.

[114] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–14. USENIX Association, 2004.

[115] M. Cremonini and M. Riccardi. The Dorothy Project: An Open Botnet Analysis Framework for Automatic Tracking and Activity Visualization. In *European Conference on Computer Network Defense (EC2ND)*, pages 52–54, 2009.

[116] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran. A foray into Conficker's logic and rendezvous points. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'09, pages 1–9. USENIX Association, 2009.

[117] Pedram Amini. Kraken Botnet Infiltration. `http://dvlabs.tippingpoint.com/blog/2008/04/28/kraken-botnet-infiltration`, April 2008. [Online; Last accessed January-7th-2014].

[118] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna. Analysis of a Botnet Takeover. *Security Privacy, IEEE*, 9(1):64–72, 2011.

[119] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security (CCS '09)*, pages 621–634. ACM, 2009.

[120] A.W. Jackson, D. Lapsley, C. Jones, M. Zatko, C. Golubitsky, and W.T. Strayer. SLINGbot: A System for Live Investigation of Next Generation Botnets. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 313–318, 2009.

[121] Chia Yuan Cho, Domagoj Babi ć, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 426–439. ACM, 2010.

[122] P. Bacher, T. Holz, M. Kotter, and G. Wicherski. Know your Enemy: Tracking Botnets. `http://www.honeynet.org/papers/bots`, October 2008. [Online; Last accessed January-7th-2014].

[123] Sergey Golovanov and Vyacheslav Rusakov. TDSS in deep analysis. `http://www.securelist.com/en/analysis/204792131/TDSS`, 1988. [Online; Last accessed January-7th-2014].

[124] Jose Nazario. Twitter-based Botnet Command Channel. http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/, August 2009. [Online; Last accessed January-7th-2014].

[125] Sergey Golovanov and Vyacheslav Rusakov. Mobile Malware Evolution: Part 6. http://www.securelist.com/en/analysis/204792283/Mobile_Malware_Evolution_Part_6, 2013. [Online; Last accessed January-7th-2014].

[126] Denis Maslennikov. ZeuS-in-the-Mobile â Facts and Theories. http://www.securelist.com/en/analysis/204792194/ZeuS_in_the_Mobile_Facts_and_Theories, 2011. [Online; Last accessed January-7th-2014].

[127] F-Secure Labs. Threat report H1 2013. http://www.f-secure.com/static/doc/labs_global/Research/Threat_Report_H1_2013.pdf, 2013. [Online; Last accessed January-7th-2014].

[128] Roman Unuchek. Obad.a Trojan now being distributed via mobile botnets. http://www.securelist.com/en/blog/8131/Obad_a_Trojan_now_being_distributed_via_mobile_botnets, 2013. [Online; Last accessed January-7th-2014].

[129] I. Arce and E. Levy. An analysis of the slapper worm. *IEEE Security & Privacy Magazine*, 1(1):82–87, January 2003.

[130] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9. USENIX Association, 2008.

[131] J. Calvet, C.R. Davis, and P.-M. Bureau. Malware authors don't learn and that's good! In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 88–97, 2009.

[132] Dae il Jang, Minsoo Kim, Hyun chul Jung, and Bong-Nam Noh. Analysis of HTTP2P botnet: case study waledac. In *9th Malaysia International Conference on Communications (MICC)*, pages 409–412, 2009.

[133] Seungwon Shin and Guofei Gu. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 151–160, New York, NY, USA, 2010. ACM.

[134] Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage. The heisenbot uncertainty problem: challenges in separating bots from chaff. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9. USENIX Association, 2008.

[135] Brent ByungHoon Kang, Eric Chan-Tin, Christopher P. Lee, James Tyra, Hun Jeong Kang, Chris Nunnery, Zachariah Wadler, Greg Sinclair, Nicholas Hopper, David Dagon, and Yongdae Kim. Towards complete node enumeration in a peer-to-peer botnet. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 23–34. ACM, 2009.

[136] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My botnet is bigger than yours (maybe better than yours): why size estimates remain challenging. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, 2007.

[137] Gao Jian, YiXian Yang, KangFeng Zheng, and ZhengMing Hu. Research of an Innovative P2P-Based Botnet. In *MVHI '10: Proceedings of the 2010 International Conference on Machine Vision and Human-machine Interface*, pages 214–218, 2010.

[138] Ralf Hund, Matthias Hamann, and Thorsten Holz. Towards Next-Generation Botnets. In *Proceedings of the 2008 European Conference on Computer Network Defense*, pages 33–40, 2008.

[139] Guenther Starnberger, Christopher Kruegel, and Engin Kirda. Overbot: a botnet protocol based on Kademlia. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, SecureComm '08, pages 1–9, 2008.

[140] Antonio Nappa, Aristide Fattori, Marco Balduzzi, Matteo Dell'Amico, and Lorenzo Cavallaro. Take a deep breath: a stealthy resilient and cost-effective botnet using Skype. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'10, pages 81–100. Springer-Verlag, 2010.

[141] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. Honeypot detection in advanced botnet attacks. *Int. J. Inf. Comput. Secur.*, 4:30–51, February 2010.

[142] Zhichun Li, Anup Goyal, Yan Chen, and Vern Paxson. Automating analysis of large-scale botnet probing events. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 11–22, 2009.

[143] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association.

[144] David Dagon, Cliff Changchun Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'06)*, 2006.

[145] Runheng Li, Liang Gan, and Yan Jia. Propagation Model for Botnet Based on Conficker Monitoring. In *Information Science and Engineering (ISISE), 2009 Second International Symposium on*, pages 185–190, 2009.

[146] David Harley, Robert S. Vibert, Ken Bechtel, Michael Blanchard, Henk Diemer, Andrew Lee, Igor Muttik, and Bojan Zdrnja. *AVIEN Malware Defense Guide for the Enterprise*. Elsevier, 2007.

[147] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Berkeley, CA, USA, 2007. USENIX Association.

[148] Michalis Polychronakis, Panayiotis Mavrommatis, and Niels Provos. Ghost turns zombie: exploring the life cycle of web-based malware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 11:1–11:8, Berkeley, CA, USA, 2008. USENIX Association.

[149] Seungwon Shin, Raymond Lin, and Guofei Gu. Cross-Analysis of Botnet Victims: New Insights and Implications. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID'11))*, September 2011.

[150] Christopher Jordan, Alice Chang, and Kun Luo. Network Malware Capture. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 293–296, 2009.

[151] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: detecting malware infection through IDS-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 167–182, 2007.

[152] Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. Defending Browsers against Drive-by-Downloads: Mitigating Heap-Spraying Code Injection Attacks. In *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '09, pages 88–106, Berlin, Heidelberg, 2009. Springer-Verlag.

[153] Basil AsSadhan, José M. F. Moura, David Lapsley, Christine Jones, and W. Timothy Strayer. Detecting Botnets Using Command and Control Traffic. In *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, NCA '09, pages 156–162, 2009.

[154] Van-Hau Pham and Marc Dacier. Honeypot Traces Forensics: The Observation Viewpoint Matters. In *Proceedings of the 2009 Third International Conference on Network and System Security*, NSS '09, pages 365–372, 2009.

[155] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, and Others. BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, pages 139–154, 2008.

[156] M. B. Priestley. *Spectral Analysis and Time Series*, page 407. Academic Press, 1982.

[157] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 232–249, Berlin, Heidelberg, 2009. Springer-Verlag.

[158] Xiaocong Yu, Xiaomei Dong, Ge Yu, Yuhai Qin, and Dejun Yue. Data-Adaptive Clustering Analysis for Online Botnet Detection. In *Third International Joint Conference on Computational Science and Optimization (CSO)*, volume 1, pages 456–460, May 2010.

[159] Su Chang and Thomas E. Daniels. P2P botnet detection using behavior clustering & statistical tests. In *AISec '09: Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pages 23–30, 2009.

[160] W. Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet Detection Based on Network Behavior. In *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 1–24. Springer US, 2008.

[161] Yuanyuan Zeng, Xin Hu, and Kang G. Shin. Detection of botnets using combined host- and network-level information. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 291–300, June 2010.

[162] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. EFFORT: Efficient and Effective Bot Malware Detection. In *Proceedings of the 31th Annual IEEE Conference on Computer Communications (INFOCOM'12)*, 2012.

[163] SeungGoo Ji, ChaeTae Im, MiJoo Kim, and HyunCheol Jeong. Botnet Detection and Response Architecture for Offering Secure Internet Services. In *International Conference on Security Technology (SECTECH '08)*, pages 101–104, 2008.

[164] Claudio Mazzariello. IRC Traffic Analysis for Botnet Detection. In *Proceedings of the Fourth International Conference on Information Assurance and Security*, pages 318–323, 2008.

[165] Wei Wang, Binxing Fang, Zhaoxin Zhang, and Chao Li. A Novel Approach to Detect IRC-Based Botnets. In *Proceedings of the 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 1, pages 408–411, 2009.

[166] Jan Goebel and Thorsten Holz. Rishi: identify bot contaminated hosts by IRC nickname evaluation. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007.

[167] Guofei Gu, V. Yegneswaran, P. Porras, J. Stoll, and Wenke Lee. Active Botnet Probing to Identify Obscure Command and Control Channels. In *Annual Computer Security Applications Conference (ACSAC)*, pages 241–253, 2009.

[168] James R. Binkley. An Algorithm for Anomaly-based Botnet Detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pages 43–48, 2006.

[169] Chia-Mei Chen, Ya-Hui Ou, and Yu-Chou Tsai. Web botnet detection based on flow information. In *International Computer Symposium (ICS)*, pages 381–384, 2010.

[170] Jae-Seo Lee, HyunCheol Jeong, Jun-Hyung Park, Minsoo Kim, and Bong-Nam Noh. The Activity Analysis of Malicious HTTP-Based Botnets Using Degree of Periodic Repeatability. In *Proceedings of the 2008 International Conference on Security Technology*, SECTECH '08, pages 83–86, 2008.

[171] Jian Kang, Jun-Yao Zhang, Qiang Li, and Zhuo Li. Detecting New P2P Botnet with Multi-chart CUSUM. In *International Conference on Networks Security, Wireless Communications and Trusted Computing (NSWCTC)*, volume 1, pages 688–691, 2009.

[172] Jian Kang and Yuan-Zhang Song. Detecting New Decentralized Botnet Based on Kalman Filter and Multi-chart CUSUM Amplification. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 1, pages 7–10, 2010.

[173] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Peer to peer botnet detection for cyber-security: a data mining approach. In *Proceedings of the 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead*, CSIIRW '08, pages 1–2, 2008.

[174] Wen-Hwa Liao and Chia-Ching Chang. Peer to Peer Botnet Detection Using Data Mining Scheme. In *International Conference on Internet Technology and Applications*, pages 1–4, 2010.

[175] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. BotGrep: finding P2P bots with structured graph analysis. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 95–110, Berkeley, CA, USA, 2010. USENIX Association.

[176] Baris Coskun, Sven Dietrich, and Nasir Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[177] Junjie Zhang, R. Perdisci, Wenke Lee, U. Sarfraz, and Xiapu Luo. Detecting stealthy P2P botnets using statistical traffic fingerprints. In *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 121–132, June 2011.

[178] Ricardo Villamarin-Salomon and Jose Carlos Brustoloni. Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic. In *5th IEEE Consumer Communications and Networking Conference (CCNC)*, pages 476–481, Jan. 2008.

[179] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 48–61, New York, NY, USA, 2010. ACM.

[180] Sandeep Yadav and A.L. Narasimha Reddy. Winning with DNS Failures: Strategies for Faster Botnet Detection. In *7th International ICST Conference on Security and Privacy in Communication Networks (SecureComm)*, London, United Kingdom, September 2011.

[181] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT)*, pages 715–720, October 2007.

[182] C.R. Davis, J.M. Fernandez, S. Neville, and J. McHugh. Sybil attacks as a mitigation strategy against the Storm botnet. In *3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pages 32–40, 2008.

[183] C.R. Davis, J.M. Fernandez, and S. Neville. Optimising sybil attacks against P2P-based botnets. In *4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 78–87, 2009.

[184] D.T. Ha, Guanhua Yan, S. Eidenbenz, and H.Q. Ngo. On the effectiveness of structural detection and defense against P2P-based botnets. In *Dependable*

Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on, pages 297–306, 29 2009-July 2 2009.

[185] A. Singh, T.-W. Ngan, P. Druschel, and D.S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses . In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006.

[186] J. Franklin, A. Perrig, V. Paxson, and S. Savage. An inquiry into the nature and causes of the wealth of Internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 375–388, New York, NY, USA, 2007. ACM.

[187] Jianwei Zhuge, Thorsten Holz, Chengyu Song, Jinpeng Guo, Xinhui Han, and Wei Zou. Studying Malicious Websites and the Underground Economy on the Chinese Web. In *2008 Workshop on the Economics of Information Security (WEIS'08)*, 2008.

[188] J. Radianti. A Study of a Social Behavior inside the Online Black Markets. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pages 189–194, July 2010.

[189] Hanno Fallmann, Gilbert Wondracek, and Christian Platzer. Covertly probing underground economy marketplaces. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'10, pages 101–110, Berlin, Heidelberg, 2010. Springer-Verlag.

[190] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. An analysis of underground forums. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 71–80, New York, NY, USA, 2011. ACM.

[191] David Molnar, Serge Egelman, and Nicolas Christin. This is your data on drugs: lessons computer security can learn from the drug war. In *Proceedings of the 2010 workshop on New security paradigms*, NSPW '10, pages 143–149, New York, NY, USA, 2010. ACM.

[192] U.S. Department of Justice. DEA Major Information Systems. `http://www.justice.gov/dea/foia/stride.html`, December 1998. [Online; accessed July-25th-2013].

[193] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning more about the underground economy: a case-study of keyloggers and dropzones. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 1–18, Berlin, Heidelberg, 2009. Springer-Verlag.

[194] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM.

[195] Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Márk Felegyhézi, Chris Grier, Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. Click Trajectories: End-to-End Analysis of the Spam Value Chain. *Security and Privacy, IEEE Symposium on*, pages 431–446, 2011.

[196] Chris Kanich, Nicholas Weaver, Damon McCoy, Tristan Halvorson, Christian Kreibich, Kirill Levchenko, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. Show Me the Money: Characterizing Spam-advertised Revenue. In *Proceedings of the USENIX Security Symposium*, San Francisco, CA, August 2011.

[197] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service. Attack and Defense Mechanisms*. Prentice Hall, 2004.

[198] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, 2004.

[199] Fang Yu, Yinglian Xie, and Qifa Ke. SBotMiner: large scale search bot detection. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 421–430, 2010.

[200] Zhenhai Duan, Peng Chen, F. Sanchez, Yingfei Dong, M. Stephenson, and J. Barker. Detecting Spam Zombies by Monitoring Outgoing Messages. In *INFOCOM 2009*, pages 1764–1772, 2009.

[201] Yinglian Xie, Fang Yu, Kannan Achan, Rina Panigrahy, Geoff Hulten, and Ivan Osipkov. Spamming botnets: signatures and characteristics. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 171–182, 2008.

[202] Yao Zhao, Yinglian Xie, Fang Yu, Qifa Ke, Yuan Yu, Yan Chen, and Eliot Gillum. BotGraph: large scale spamming botnet detection. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 321–334, Berkeley, CA, USA, 2009. USENIX Association.

[203] B. Al-Duwairi and G. Manimaran. JUST-Google: A Search Engine-Based Defense Against Botnet-Based DDoS Attacks. In *IEEE International Conference on Communications (ICC)*, pages 1–5, 2009.

[204] M. Patrick Collins, Timothy J. Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. Using uncleanliness to predict future botnet addresses. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 93–104, 2007.

[205] Wei Yan, Zheng Zhang, and N. Ansari. Revealing Packed Malware. *Security Privacy, IEEE*, 6(5):65–69, Sept.-Oct. 2008.

[206] Tom Brosch and Maik Morgenstern. Runtime Packers: The Hidden Problem? `http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf`, 2006. [Online; Last accessed January-7th-2014].

[207] Jon Oberheide, Evan Cooke, and Farnam Jahanian. CloudAV: N-version antivirus in the network cloud. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 91–106, Berkeley, CA, USA, 2008. USENIX Association.

[208] The Honeynet Project. Know Your Enemy: Fast-Flux Service Networks. `http://www.honeynet.org/papers/ff`, July 2007. [Online; Last accessed January-7th-2014].

[209] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. FluXOR: Detecting and Monitoring Fast-Flux Service Networks. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 186–206, 2008.

[210] Thorsten Holz, Christian Gorecki, Felix Freiling, and Konrad Rieck. Measuring and Detecting Fast-Flux Service Networks. 2008.

[211] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pages 24–31, 2008.

[212] Alper Caglayan, Mike Toothaker, Dan Drapaeau, Dustin Burke, and Gerry Eaton. Behavioral analysis of fast flux service networks. In *CSIIRW '09: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, 2009.

[213] Alper Caglayan, Mike Toothaker, Dan Drapeau, Dustin Burke, and Gerry Eaton. Real-Time Detection of Fast Flux Service Networks. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 285–292, 2009.

[214] R. Perdisci, I. Corona, D. Dagon, and Wenke Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *Annual Computer Security Applications Conference (ACSAC)*, pages 311–320, 2009.

[215] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE : Finding malicious domains using passive DNS analysis. In *NDSS11, 18th Annual Network & Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*, San Diego, UNITED STATES, 02 2011.

[216] Manos Antonakakis, Roberto Perdisci, Wenke Lee, II Nikolaos Vasiloglou, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association.

[217] Ping Wang, Lei Wu, Baber Aslam, and Cliff C. Zou. *A Systematic Study on Peer-to-Peer Botnets*. IEEE, August 2009.

[218] Felix Leder, Tillmann Werner, and Peter Martini. Proactive Botnet Countermeasures An Offensive Approach. In *1st Conference on Cyber Warfare (CCDECEO)*, June 2009.

[219] ENISA. Botnets: Detection Measurement Disinfection & Defence. Technical report, European Network and Information Security Agency (ENISA), 2011.

[220] Juniper. 2011 Mobile Threats Report. Technical report, Juniper Networks, Feb 2012.

[221] Lei Zhang, Shui Yu, Di Wu, and P. Watters. A Survey on Latest Botnet Attack and Defense. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 53–60, 2011.

[222] M. Feily, A. Shahrestani, and S. Ramadass. A Survey of Botnet and Botnet Detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 268–273, 2009.

[223] Liping Feng, Xiaofeng Liao, Qi Han, and Lipeng Song. Modeling and Analysis of Peer-to-Peer Botnets. *Discrete Dynamics in Nature and Society*, pages 1–18, 2012.

[224] K. Muthumanickam and E. Ilavarasan. P2P Botnet detection: Combined host- and network-level analysis. In *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*, pages 1–5, 2012.

[225] Ihsan Ullah, Naveed Khan, and Hatim A. Aboalsamh. Survey on botnet: Its architecture detection prevention and mitigation. In *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on*, pages 660–665, 2013.

[226] Rafael A. Rodríguez-Gómez, Gabriel Maciá-Fernández, and Pedro García-Teodoro. Survey and Taxonomy of Botnet Research through Life-cycle. *ACM Computing Surveys*, 45(4):1–39, 2013.

[227] The Pirate Bay Web Page. `http://thepiratebay.sx/`. [Online; Last accessed January-7th-2014].

[228] Torrentz Web Page. `http://torrentz.eu/`. [Online; Last accessed January-7th-2014].

[229] The BitTorrent Protocol Specification. `http://bittorrent.org/beps/bep_0003.html`. [Online; Last accessed January-7th-2014].

[230] Mainline DHT Implementation. `http://bittorrent.org/beps/bep_0005.html`. [Online; Last accessed January-7th-2014].

[231] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, 2002.

[232] K. Junemann, P. Andelfinger, J. Dinger, and H. Hartenstein. BitMON: A Tool for Automated Monitoring of the BitTorrent DHT . In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–2, 2010.

[233] Masood Khosroshahy, Mustafa K. Mehmet Ali, and Dongyu Qiu. The SIC botnet lifecycle model: A step beyond traditional epidemiological models. *Comput. Netw.*, 57(2):404–421, February 2013.

[234] Matteo Varvello and Moritz Steiner. Traffic localization for DHT-based BitTorrent networks. In *Proceedings of the 10th international IFIP TC 6 conference on Networking*, NETWORKING'11, pages 40–53, 2011.

[235] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Long term study of peer behavior in the KAD DHT. *IEEE/ACM Trans. Netw.*, 17(5):1371–1384, October 2009.

[236] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 189–202. ACM, 2006.

[237] GeoIP country database reader in Python. `https://code.google.com/p/python-geoip/`. [Online; Last accessed January-7th-2014].

[238] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. EFFORT: Efficient and Effective Bot Malware Detection. In *Proceedings of the 31th Annual IEEE Conference on Computer Communications (INFOCOM'12)*, 2012.

[239] Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. Detecting stealthy P2P botnets using statistical traffic fingerprints. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 121–132. IEEE, 2011.

Apéndices

Appendix A

Thesis Summary

In this appendix we present an extended abstract of the present thesis. Here we expose briefly the two parts of our work: the first one in the field of network traffic classification, and the second about the study and detection of botnets. Both parts are intended to increase the security of P2P networks, and thus this is the type of traffic considered.

The remainder of the appendix is organized as follows. Firstly, in Section A.1 we motivate the importance of the security in P2P networks and highlight one of the most relevant threats nowadays: botnets. After that, in Section A.2 we clarify the objectives of the present work and relate its main contributions in Section A.3. Finally, Section A.4 constitutes the main part of this appendix, and here we briefly describe each contribution of our work. In Appendix B we draw the main conclusions of this work and point out some open issues for future research.

A.1. Motivation

These days are known as the communications era and one of the main reasons for it is the penetration and current use of the Internet. Nowadays, the Internet is present in frequent daily interactions of a great part of the world. In fact, the *Network of networks* provides service for billions of users all around the globe, from their houses with a computer, a smartphone, or even from smart appliances. In the Internet there exist several services that allow users to communicate with others without effort. These are well known services like web, e-mail or services to share resources through P2P networks, among others.

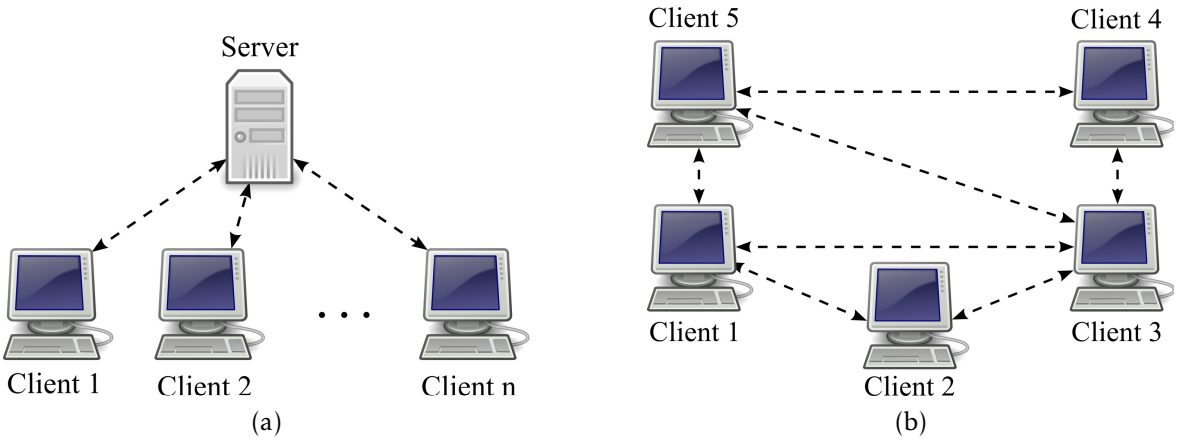


Figure A.1: Scheme of the (a) client-server and (b) P2P models.

The first applications in the Internet were based on the client-server paradigm, still used in very common services like the web or e-mail (see Figure A.1(a)). In this communication model, an application installed in a client interacts with other one that is running on a server. The client is the entity in charge of starting a communication with the server by sending a *request*, while the latter should wait for new requests. After that, the server is in charge of processing the request sent by the client and send a *response*.

One of the main advantages of this paradigm is that it is very easy to implement the associated communications. However, the most important disadvantage is its reduced scalability. It is hard to maintain the same service level offered by a server specially when the number of clients grows substantially.

Subsequently to the appearance of the client-server paradigm, and motivated by the increment in the capacity of the users bandwidth, the P2P model was created (see Figure A.1(b)). In this new model, each entity has the capacity of acting as a client and as a server as well. This way, it is possible for an entity to start communications with others and also to receive incoming requests. Therefore, we might say that all the participants in this model play the same role and, as a consequence, they are named *peers*.

In this new communication model it is possible to manage the scalability problem in a more efficient way than in the case of the client-server model. The P2P model allows to distribute the execution of a particular service between multiple peers of the network. However, the network management becomes now a more complex task. For example, it is necessary to establish mechanisms to identify where the provided services can be found, or to design specific protocols for the communication between peers, etc. Examples of P2P networks are eDonkey, BitTorrent, Spotify or Skype, among others.

Traffic in P2P networks represents a considerable percentage of the current world's bandwidth. In 2009, it supposed between 43% and 70% of all the Internet bandwidth [2]. And although the percentages related to different P2P networks can be different, we can not deny that the network traffic volume derived from the usage of P2P applications supposes a substantial percentage of the world's bandwidth.

However, this new communication model presents a relevant disadvantage, mainly due to the appearance of new vulnerabilities and security threats. The own design of the P2P model implies that there exists no central server in charge of uploading, saving and checking the authenticity of the shared resources. As a consequence, there are no mechanisms to control the contents shared in the network. This way, P2P networks are ideal to propagate malware. In addition, nodes in these networks are really good candidates to be recruited for the execution of attacks like DDoS, due to their considerable upload and download capacity [3].

Network security is a specially relevant topic in the communication era, very recurrent in research and it is frequent to find news related with it. But, is it really so important? What is its relevance? A piece of information that could shed some light on the importance of this problem is a study published in 2011 by Norton Symantec [4]:

For the first time a Norton study calculates the cost of global cybercrime: $114 billion annually. Based on the value victims surveyed placed on time lost due to their cybercrime experiences, an additional $274 billion was lost. With 431 million adult victims globally in the past year and at an annual price of $388 billion globally based on financial losses and time lost, cybercrime costs the world significantly more than the global black market in marijuana, cocaine and heroin combined ($288 billion).

Motivated by the growing relevance of network security in general, the main objective of this work is *to improve the security of P2P networks*. To do this, we propose an approach in two steps. The first step is to design efficient and effective mechanisms *to identify P2P network traffic*. Network traffic classification [5] is the automatic process for network traffic being categorized in different classes in function of several parameters, for example, the communication port, the packet headers, flow features, etc. It is of great importance that the ISPs are able to classify their traffic in order to apply security policies, or even anomaly detection/response systems depending on the identified traffic. For this reason, traffic classification is nowadays a very popular topic in the field of networks, and more precisely between the research community of this field.

Once the traffic classification problem has been addressed, the second step of the proposed approach in this work is to apply particular defense measures against attacks to the security in P2P networks. After a study of the current cibersecurity attacks in P2P networks it is shown that one of the main current threats are the zombie networks or *botnets*. The term botnet is used to define a network of infected

machines, called *bots*, which are controlled by a human operator named *botmaster*. Bots are used in a wide range of malicious and harmful activities against systems and services, including DoS attacks, spam and malware distribution, phishing and click fraud [6] [7].

The growing danger related to the botnets has been pointed out by several authors in different works, as in the case of Vinton Cerf, one of the "fathers of the Internet", who estimated that 100 and 150 of the 600 millions of hosts connected to the Internet were part of a botnet [8]; or a study of Symantec in 2007 [9], in which they detected a mean of 61,940 active bots per day; or a later study of McAfee in 2009 [10], in which they discovered almost 12 millions of new IP addresses acting as bots, a remarkable increment with respect to the previous study of Symantec.

This problem requires a detailed study of the current bibliography and a specific solution to the threats still unsolved in the field of botnets. This is the context and the aim of the present work.

A.2. Objectives

The main purpose of this thesis is to improve the security of P2P networks. This general purpose is divided into two specific objectives: (*i*) identification of P2P network traffic, and (*ii*) study and detection of P2P botnets. In the following, we detail the associated tasks to each of these objectives:

I. *Identification of P2P network traffic*

- To design an efficient and effective P2P traffic identification system.
- To physically implement the designed system.
- To evaluate its correct operation by means of realistic experiments.

II. *Study and detection of botnets*

- To study the state of the art of research in the field of botnets.
- To organize, following an intuitive approach, the studied research works.
- To design a detection scheme against P2P parasite botnets, a threat specially relevant and harmful, yet without particular defense proposals.
- To evaluate the proposed detection scheme in a realistic scenario.

A.3. Main Contributions

The main contributions of this thesis can be summarized in the following items:

1. We present a brief bibliographic review of the network traffic classification field.
2. We contribute with two heuristic-based detection algorithms for eDonkey traffic. One of them is able to detect flows of this protocol, WCFD, and the second one is intended to detect nodes that generate eDonkey traffic, URND.
3. We design and implement a network traffic classification system based on hidden Markov models that can be easily used to detect multiple different protocols. We show how to detect eDonkey following this approach as a case study.
4. We also design a taxonomy based on the life-cycle of botnets which allows to group in an organized way the research carried out in this field.
5. We contribute with a review of the most relevant works in the last years in the field of botnets. We organize these works following the previous taxonomy.
6. We introduce a detection system for one of the most important new threats related to the field of botnets: P2P parasite botnets. This detection approach represents an innovation regarding previous botnet detection schemes because it is based on the temporal evolution of shared resources between bots instead of on network communications among them.

A.3.1. Publications

In the following, we indicate the publications directly related with the main topic of the present thesis:

International Journals

1. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, and P. García-Teodoro. 2013. "Survey and taxonomy of *botnet* research through life-cycle". *ACM Comput. Surv.* 45, 4, Article 45 (August 2013), 33 pages.
2. *Submitted* – > **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, P. García-Teodoro, M. Steiner, and D. Balzarotti. 2013. "Resource monitoring for the detection of parasite P2P botnets". *Computer Networks*.

3. *Selected to be submitted* – > **R. A. Rodríguez-Gómez**, G. Maciá-Fernández, and P. García-Teodoro. 2013. "Analysis and monitoring of resources shared in BitTorrent Network". *Transactions on Emerging Telecommunications Technologies*.

International Conferences

4. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Stochastic Traffic Identification for Security Management: eDonkey Protocol as a Case Study". *7th Network and System Security* (NSS 2013), pp. 1-13, 2013.

5. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Analysis of *Botnets* through Life-cycle". Proceedings of *SECRYPT*, pp. 257-262, 2011.

6. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "New Heuristics for eDonkey Node and Flow Detection". *P2P Systems* (AP2PS), pp. 90-95, 2011.

National Conferences

7. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Monitorización y Análisis de los Recursos Compartidos en la Red BitTorrent". *XI Jornadas de Ingeniería Telemática* (JITEL 2013), pp. 243-248.

8. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Acceso a servicios basado en modelado de Markov: eDonkey como caso de estudio". *Reunión Española sobre Criptología y Seguridad de la Información* (RECSI 2012), pp. 51-56.

9. **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and P. García-Teodoro. "Nuevas heurísticas para la detección de nodos y flujos eDonkey". *X Jornadas de Ingeniería Telemática* (JITEL 2011), pp. 276-283.

Other publications which are not directly related with the main work of this thesis but which are part of the work carried out in the doctoral period are:

International Journals

a) G. Maciá-Fernández, Y. Wang, **R. A. Rodríguez-Gómez** and A. Kuzmanovic. "Extracting User Web Browsing Patterns from Non-Content Network Traces: The Online Advertising Case Study", *Computer Networks*, Vol. 56. 2012, pp. 598-614.

b) G. Maciá-Fernández, **R. A. Rodríguez-Gómez** and J. E. Díaz-Verdejo. "Defense techniques for low-rate DoS attacks against application servers", *Computer Networks*, Vol. 54, October, 2010, pp. 2711-2727.

International Conferences

c) L. Sánchez-Casado, **R. A. Rodríguez-Gómez**, R. Magán-Carrión and G. Maciá-Fernández. "NETA: Evaluating the effects of NETwork Attacks. MANETs as a case study". *Advances in Security of Information and Communication Networks*, (SecNet 2013), pp. 1-10.

d) G. Maciá-Fernández, Y. Wang, **R. A. Rodríguez-Gómez** and A. Kuzmanovic. "ISP-enabled behavioral ad targeting without deep packet inspection". Proceedings of the *29th conference on Information communications*, (INFOCOM 2010), pp. 1-9.

National Conferences

e) L. Sánchez-Casado, **R. A. Rodríguez-Gómez**, R. Magán-Carrión and G. Maciá-Fernández. "NETA: un Framework para Simular y Evaluar Ataques en Redes Heterogéneas. MANETs como Caso de Estudio". *XI Jornadas de Ingeniería Telemática* (JITEL 2013), pp. 501-506.

f) **R. A. Rodríguez-Gómez**, G. Maciá-Fernández and J. E. Díaz-Verdejo. "Defensas frente a ataques DoS a baja tasa contra servidores basadas en políticas de gestión de colas". *VIII Jornadas de Ingeniería Telemática* (JITEL 2009), pp. 46-53.

Books

g) G. Maciá Fernández, R. Magán Carrión, **R. A. Rodríguez Gómez**, L. Sánchez Casado: "Sistemas y Servicios Telemáticos". 2013. Ed. Avicam. ISBN: 978-84-941781-6-0.

A.4. Extended Abstract

This work, in the same way as its objectives, has been divided into two main parts, both intended to improve the security of P2P networks.

The first one is focused on traffic classification techniques. In this field we have described, in Section A.4.1, two heuristic-based detection algorithms aimed to identify eDonkey traffic. One of them is intended to detect eDonkey flows (WCFD) and

the other nodes that generate traffic of the mentioned protocol (URND) (see publications 6 and 9). In Section A.4.2, we detail a classification system based on HMM (see publications 4 and 8). The main difference with respect to previous heuristic detection scheme is that it is quite easy to configure the Markov system to detect multiple different protocols.

The second part of this work is related to one of the most critic topics of our days in cibersecurity, *i.e.*botnets. Firstly, in Section A.4.3, we present a new taxonomy of botnets based on their life-cycle, and based on it we review the bibliography on this topic (see publications 1 and 5).

One of the new threats discovered in the mentioned study of botnets is a specific type of botnets which communicate through an existent P2P network, named P2P parasite botnets. They hide their traffic inside the normal traffic of the legitimate P2P network and because of that it is hard to detect their communications. In Section A.4.4 we describe a detection system for this type of botnets based on the temporal evolution of the shared resources between bots (see publications 2 and 7).

First Part: P2P Traffic Identification

A.4.1. New Heuristics for Identification of eDonkey Protocol (publications 6 and 9)

Two detection methods for identifying eDonkey traffic are proposed: *Up Rate-based Node Detection*, (URND) and *Way Changed-based Flow Detection*, (WCFD). The detection heuristics used in both methods are indicated in the following. First, the datasets of traffic used in experimentation are described.

Datasets

Four groups of network traces have been used to carry out the evaluation of the proposed detection systems. The principal features of these traces are exposed now:

- *Controlled environment traces (EC).* The traffic generated by 5 users during 72 hours were collected. They limited the up-rate of aMule to 30kB/s. All of them used their PCs and Internet connection without restrictions. Every user generated around 19,000 connections of the eDonkey protocol and more than 7,000 of other protocols like DNS, HTTP, SSH and SMTP.

- *eDonkey traffic traces (eD).* This trace is composed of the eDonkey traffic generated by a node during 45 days. These network traces contain 240,851 TCP flows and 7,003 UDP flows, all of them belonging to the eDonkey protocol. Among the total amount of TCP flows, 22,409 are obfuscated.

- *HTTP server traces (SW).* This collection of traces represents the traffic generated and received by an HTTP server from an European University during 7 days. The server is Apache version 2.2.0 and receives a mean of 8,971 connections per day.

- *University trunk traces (TU).* All the traffic of a trunk from a Middle East University during 48 hours composes these traces. There are around 73,000 IPs, 300 millions of packets transmitted, and the most common protocols that appear are: BitTorrent, HTTP, DNS, SSL, and FTP.

eDonkey Flows Detection: WCFD

In client-server applications, servers usually send the majority of the data after a connection is started by a client. This behavior is reverse to that of eDonkey protocol, and this is the reason because we propose to use the next hypothesis to detect eDonkey flows:

Hypothesis 1: *eDonkey flows are those in which clients who begin the connection send substantially more information than they receive.*

Note that this heuristic is only valid for file sharing flows, and not in the case of signaling flows. File sharing flows are specially relevant because they are the principal responsible of congesting the bandwidth of a network.

Experimental Results for WCFD

First of all, to execute the eDonkey flow detection method it is necessary to determine the maximum difference allowed between the number of the sent and received bytes to consider the evaluated node as non-generator of eDonkey traffic; this value is named $Thres_B$. A study of $Thres_B$ reveals that there exists a wide range (between 5 and 25kB) for this threshold in which the results provided by the method are very similar. Finally, the threshold selected is 10kB.

Using WCFD, we have been able to identify 28,016 eDonkey flows of a total of 37,089 file sharing flows of eDonkey protocol in the used network traces. Yet, there exists a considerable false negative rate: 22.47 %. This is mainly due to two factors: low ID of eDonkey nodes in some of the ends, and service without an intermediate

Table A.1: FPR of flow detection method in TU and SH traces.

Protocol	FPR	Detected flows	Total flows
BitTorrent	2.56%	854	33,304
HTTP	1.691%	50,795	3,003,161
FTP	1.423%	35	2,460
SSL	1.244%	2,808	225,685
IRC	0.213%	7	3,281
Oscar	0.079%	2	2,528
DNS	0.001%	8	1,508,413
Mail_POP	0.000%	0	5,208
All	1.139%	54,509	4,784,040

close of connection. Therefore, the proposed detection method is able to detect file sharing flows of eDonkey protocol between high ID nodes and with an intermediate close of connection.

Moreover, we have used SH and TU traces to estimate the false positive rate of the WCFD approach. As show in Table A.1. The principal contribution in false positive rate corresponds to BitTorrent protocol. This protocol is also used in P2P file sharing applications. However, a principal difference with eDonkey is that BitTorrent flows are bidirectional, so both peers send information to the other in the same flow and it can occur that clients that begin a connection send more than they receive. In the same manner, FTP flows were detected because clients frequently send more data than servers do.

Finally, the false positive rate associated to the HTTP protocol is remarkable, because it shows a 1.69% of FPR. After a detailed study, we conclude that the false positives are mainly caused by three factors: *(i)* high length of cookies and URLs in HTTP GET messages, *(ii)* very short server responses, *e.g.*, 304 (Not Modified), and *(iii)* HTTP POST sending a big amount of data.

eDonkey Nodes Detection: URND

The second heuristic detection method here is based on the assumption that users of P2P file sharing applications usually limit their upload-rate. This is due to the fact that these applications consume the majority of the upload bandwidth and, consequently, users without a limitation in the upload-rate suffer a decrease in the quality of their normal web browsing. In conclusion, the proposed detection hypothesis is defined as follow:

Hypothesis 2: *Nodes generators of eDonkey traffic are those with a quasi-constant upload-rate.*

To apply the previous hypothesis it is necessary to detect a quasi-constant level in the upload-rate of a node. We take as a reference the work of J. Ramirez et al. [54], in which they use the Kullback-Leibler (KL) divergence to detect voice activity in audio signals. The KL divergence can be described as an indicator of the similarity between two probability distributions. In the case of two Gaussian distributions p_L and p_R in its symmetric form is defined as:

$$\rho_{L,R} = \frac{1}{2}\left[\frac{\sigma_L^2}{\sigma_R^2} + \frac{\sigma_R^2}{\sigma_L^2} - 2 + (\mu_L - \mu_R)^2\left(\frac{1}{\sigma_L^2} + \frac{1}{\sigma_R^2}\right)\right] \tag{A.1}$$

The proposed detection method can be described as follows. Firstly, upload-rate values are calculated every t seconds. A median filter [55] of length N is applied to the resulting values.

Secondly, the means and variances of the filter values are calculated by means of two consecutive windows (v_I y v_D) of length N. The Gaussian distributions represented by these means and variances should be very similar if there exists a quasi-constant upload-rate. Therefore, the KL "distance" (Equation (A.1)) computed from these means and variances should be minor than a threshold, $Thres_{KL}$, to represent a constant upload-rate. If a constant rate is detected, our method will classify the corresponding node as an eDonkey traffic generator.

Experimental Results for URND

The second detection hypothesis is validated with traces of eDonkey nodes with an upload-rate limit. As expected, during the 72 hours of traces there exists a quasi-constant behavior around 30kB/s, the limit fixed in the experiment.

In Figure A.2, the upload-rate of a user during the first hour of the capture is shown. The upload-rate presents two well defined sections: near to zero, and around 30kB/s. During the first 30 minutes the upload-rate is near to zero because only few nodes know the existence of the monitored peer. After that, we can observe the mentioned constant behavior around 30kB/s. The upload-rate median filtered is also presented in the same figure, which allows us to appreciate the suppression of values that extremely deviate from the expected ones.

The KL distance is also shown in this figure and is divided into three parts: two sections near zero, separated by a clear increase in the KL distance. KL distances near zero represent the constant upload-rate sections, and a relative maximum of the KL distance indicates a change in the upload-rate distribution. The second section of the

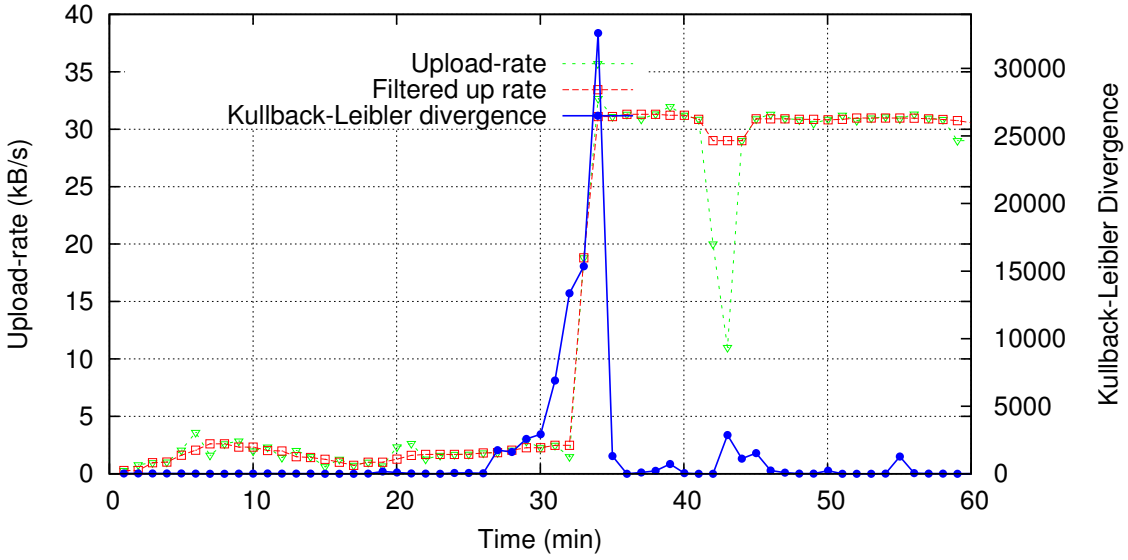


Figure A.2: Evolution of Kullback-Leibler distance in one hour of EC traces.

near zero KL distance will be identified as generated by an eDonkey node because it presents a constant upload-rate different to zero.

The execution of the URND method over these traces results in the classification of these nodes as generators of eDonkey traffic during a 86.10 % of the monitored time. The rest of the time is mainly composed of instants at which clients stop sharing with certain client and consequently their upload-rates are not constant.

Finally, the FPR has also been analyzed. We have identified an HTTP server as an eDonkey node only during 1.687 % of the total monitored time (168 hours in total).

Comparison of Obtained Classification Results

Both identification algorithms, WCFD and URND, present considerably good classifications results, but their main advantage is the simplicity of the proposed approach which allows a high computational efficient. They widely improve the detection results for file sharing protocols obtained by Moore et al. [15] with a TPR of 36,45 %. WCFD is valid for obfuscated flows unlike the work of Sen et al. en [29] and it also improves the FPR of this last work with around 5 %.

A.4.2. Detection System based on HMM (publications 4 and 8)

Here, we present a stochastic detection approach, based on the use of Markov models for classifying network traffic. In this system it is necessary to specify: *(i)*

the network traffic features that should be used as the observations for the Markov model, and *(ii)* the Markov model itself (*i.e.*, states, transitions, etc).

HMM Specifications

Model Observations: Preprocessing

Our traffic classification approach relies on a flow identification basis. Here, we consider that a flow [46] is the traffic identified by the tuple `<source-IP-address, source-port,destination-IP-address,destination-port,IP-protocol>`, where source and destination are interchangeable to allow bidirectional traffic.

Each flow is specified in terms of a sequence of packets where each packet constitutes an incoming observation. A task before introducing observations to our system is a preprocessing stage composed of three modules: *(i)* parameterization, *(ii)* normalization, and *(iii)* vector quantization.

Parameterization. The election of the vector features is a key issue and it directly influences on the discerning capability of the system. The selected features should be independent of the transmission mode for the information. This allows the detection of both plain and encrypted communications. The selected features are three:

1. *Inter-arrival time (itime):* It is defined for a packet as the difference, in seconds, between the arrival time of that packet and the arrival time of the previous one in the same flow.

2. *Payload size (psize):* It is the size, in bytes, of the information carried out by a packet.

3. *Change of direction (chdir):* This feature takes a value '1' for a given packet if it travels in the opposite direction than the previous packet in the same flow. Otherwise, it takes a value '-1'.

Normalization. The normalization module is aimed at giving the same dynamic range for the three extracted features of a packet. The selected range is [-1,1].

Vector Quantization. We finally carry out a quantization process on the obtained vectors of features for every observation (packet) in a flow. The vector quantization process used in our system is based on the K-means algorithm [60]. From the K-clustered space, each incoming feature vector will be replaced by the centroid of the nearest cluster. Summarizing, a given flow, after this preprocessing stage, will be represented by a sequence of indexes which are the observations for the HMM.

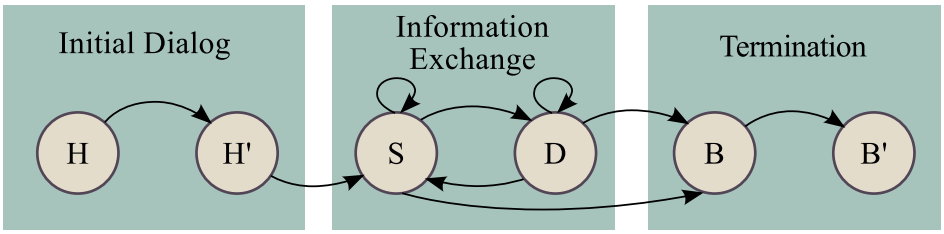


Figure A.3: HMM for Modeling eDonkey protocol

HMM Structure

The key feature of the proposed detection methodology is the use of a Markov model to describe the traffic corresponding to a given protocol/service. Although the specific definition of the parameters of the Markov model depends on the protocol to be detected, in the following we propose three stages of a generic model to represent most of the current communication services:

1. *Initial dialog:* This represents the start of the communication. Frequently in this step, the members of the communication exchange an identifier that will be used afterwards.
2. *Information exchange:* Here, the participants transmit the information that constitutes the main purpose of the communication, e.g., file transfer, web page retrieval, email sending, etc. Thus, it is expected that the bulk of the packets in the flow are transferred in this step.
3. *Termination:* After the information exchange step, it is usual that many protocols exchange some messages with the aim of finalizing the communication.

An important issue in the design of our Markov model is the variability in the number of packets (observations) of different flows, even for a same protocol/service. To consider in our model this variability we define the *information exchange* step as re-executable. Finally, note that the *initial dialog* and the *termination* are two steps executed only once in the model.

HMM for the Detection of eDonkey Flows

With the aim of exploring the proposed generic model in a case study, we have carried out an adaptation to build a detection system for the eDonkey protocol. We focus our detection in eDonkey TCP flows which represent more than 95% of the total eDonkey flows and more than 99% of the transmitted bytes.

The proposed Markov model is shown in Figure A.3. The first and the last steps are composed of two states each one and there exists only a possible transition between

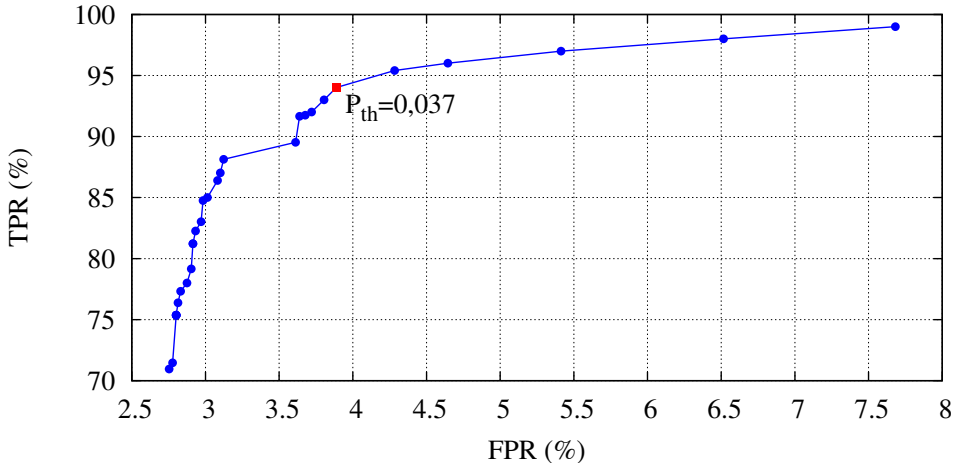


Figure A.4: ROC curve of the eDonkey detection

them. This is due to the own behavior of the eDonkey protocol. The messages exchanged during the *initial dialog* step are `Hello` (represented by state H) and `Helloanswer` (state H') messages. The messages transmitted in the *termination* step are `Startuploadrequest` (state B) and `Queuerank` (state B') messages.

Regarding the *information exchange* step we cannot define a state for every packet in the flow, as there is a variable number of packets in different flows. Now, we define two possible ways: *(i)* one for data flows (high number of packets with a high size per packet), and *(ii)* the other one for signaling flows (reduced number of packets with a low size per packet). Each one of these flows are composed of two states, one representing signaling packets (S), and the other (D) data packets. The transitions defined here allow a variable number of packets in every transfer, and all the possible combinations between these two types of packets. Finally, the transition to the *termination* step can occur also from both signaling or data states.

Detection Results

We have used K-means with a value $K = 32$ and the Euclidean distance for preprocessing the observations in experimentation. We follow a cross validation process obtaining more than 95% of true detection with around 4% of false positives.

We obtain the ROC curve of the system with a validation dataset not used in the training phase. The results can be seen in Figure A.4 and we can conclude that there exists a wide range of TPR (between 0.9 and 0.95) in which the system gives small values of FPR (from 0.036 to 0.038).

The datasets used contain both obfuscated and plain flows. Specifically, the TPR with eDonkey obfuscated flows is 96.6% and 94.2% for non-obfuscated ones. These results show the independence of the system and the encryption of the protocol.

Table A.2: False positive rates for the detection of eDonkey when other protocols are analyzed.

	Total flows	FPR %
HTTP	710,037	3.6 %
RTP	58,509	0.1 %
P2P file sharing	215,203	1.7 %

Regarding FPR in Table A.2 we show the total amount of flows and the rate of false positives yielded by our detection system. Note that our system is able to distinguish flows from other files sharing applications and from multimedia streaming giving a very low false positive rate. Finally, HTTP is also well classified as not P2P by the detection system, yielding an acceptable false positive rate.

Comparison of Obtained Classification Results

Our approach is independent of the number of packets in a flow and moreover, is able of achieving higher detection results than [43]. It also exhibits higher TPR comparing with [37] and [44].

In the work [45] the authors train the system with few flows, for example, eDonkey is trained with 109 and evaluated with 82. Instead, our training and evaluation use more than 240 thousands of eDonkey flows, reaching a higher true detection rate and a similar false positive.

It is remarkable also the detection capabilities of our approach with respect to other works in the literature [15] [29] [38] and even with respect to the proposed aproach in Chapter 3 which reached detection rates TPR and FPR of 77,53 % and 1,139 % respectively for the case of WCFD and of 86,10 % and 1,678 % for the case of URND.

Summarizing, the results obtained show that our approach is able to detect eDonkey flows with high accuracy with respect to other works in the literature.

Second Part: Study and Detection of Botnets

A.4.3. Study of Botnets through Life-cycle (publications 1 and 5)

Here we propose a taxonomy of botnet research following a new life-cycle of botnets. Based on it we present a survey of the field to provide a comprehensive overview of all these contributions. See publication 1 for more details about the survey.

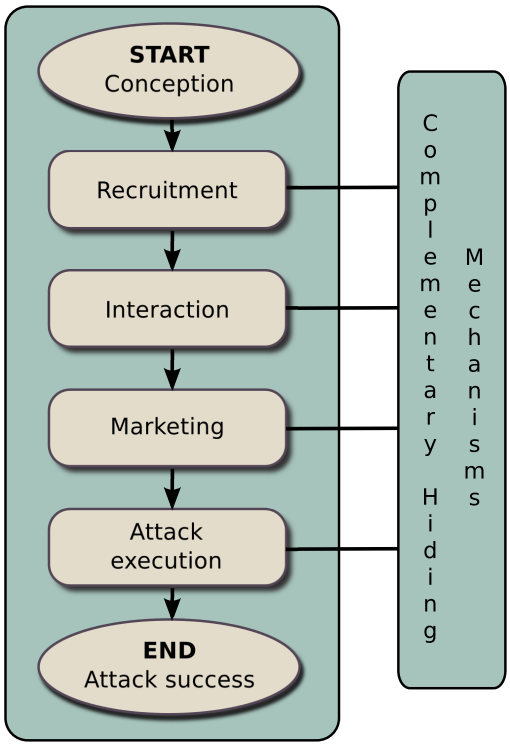


Figure A.5: Stages and complementary hiding mechanisms of botnet life-cycle.

The life-cycle proposed here is a linear sequence of stages. The end of the life-cycle, *i.e.*, attack success, is reached only after all the previous stages have been successfully carried out. Specifically, this life-cycle is compose of six stages (see Figure A.5): *conception*, *recruitment*, *interaction*, *marketing*, *attack execution* and *attack success*. There also exist a series of complementary mechanisms to the stages of the botnet life-cycle. These mechanisms are usually focused on trying to hide the botnet from security officer's eyes. In summary, we claim that ***interrupting the execution of only one stage in the botnet life-cycle makes the whole botnet useless.***

In what follows, the different stages and processes of the proposed life-cycle are briefly described.

Conception Stage

The first stage of the proposed life-cycle is the *conception* of the botnet. It can be divided into three phases or processes: *motivation*, *design* and *implementation*. First, a potential botmaster (person who manages the bots) needs a good reason to create a botnet. The motivations of a botmaster could be classified as [76]: Money, Entertainment, Ego, Cause, Entrance to social groups, and Status, which form the allegoric acronym MEECES. However, the major motivations are those related to economical profit.

Whatever the reasons of potential botmasters are, the process which follows the motivation is to design and implement the desired botnet. Here, a key decision at this point is the architecture of the botnet: *centralized*, *distributed*, or *hybrid*.

Once the botnet is conceptually conceived and designed, the last process of this stage is the own implementation of the bot code, following a traditional software development process.

Recruitment Stage

The implemented botnet software will be deployed for its operation in a real environment. For that purpose, some bots should be recruited. Indeed, the aim of the botmaster is to find the maximum number of vulnerable systems, in order to install his bot. Note that this problem is not particular of botnets, but common to many cyberattacks. In fact, the recruitment, also known as *infection*, has been widely studied in the specialized literature.

Interaction Stage

This stage refers to all the interactions performed during the botnet operation. One of the main differences between botnets and other type of malwares is the existence of communications by using C&C messages. These are of special relevance and a great amount of research papers on botnets are directly related with this aspect. This makes the botnet interaction stage a principal concern for the research community.

The processes involved in this third stage can be classified as *internal* and *external*.

Internal Interactions: are those carried out between members of the botnet, *i.e.*, from the botmaster to the bots or vice versa, or only between bots. Here, we find two processes: *registration* and *C&C communications*.

- Registration process. Registration is the process through which a compromised host becomes an effective part of the botnet.
- C&C communications. The bulk of the interactions in the botnet occur after the registration process is completed. These interactions are the C&C communications.

External Interactions: are those related to communications carried out between a member of the botnet and a non-compromised host. These usually correspond to the access to common services offered in the Internet.

Marketing Stage

At this point, the botnet has been created and it is plenty of functionality after the previous stages. Now, the botmaster needs some motivation to use it and the most common is that of earning money.

The expected economical profit is usually obtained by *(i)* selling the botnet code or, most commonly, by *(ii)* renting the botnet code or its services. In both cases, an advertisement procedure is needed, through which the malicious user announces the capabilities and the services offered by the botnet. A report from Namestnikov, Karspersky Lab [102], presents a list of prices for renting services of a botnet, *p.ej.*, *sending spam email* to a list of around a million of addresses ranges from $150 to $200.

Attack Execution Stage

The final goal of a botnet is the execution of an attack. The main feature of botnet attacks is the enormous amount of attackers that take part on them. Thus, the principal attacks launched by botnets are: Distributed denial-of-service (DDoS), spamming, phishing, or click fraud, among others.

Complementary Hiding Mechanisms

Complementary hiding mechanism are also considered as part of the proposed botnet life-cycle, although they are not defined as an stage. These are mechanisms designed for hiding the botnet and making it difficult to discover its components (bots, botmaster, C&C channels). Regretfully, even if we were able to disclose a hiding mechanism we have not still defeated the botnet. For this reason, we consider these mechanisms not as an stage, but complementary to them.

There are many possible hiding techniques studied in the literature: ciphering, polymorphism, IP spoofing, e-mail spoofing, fast-flux network, between others.

Future Research Guidelines

Despite the considerable number of proposals made and work carried out in the field of botnet detection and prevention, our study reveals that there are still certain aspects that should be investigated in greater depth. We now present the main challenges currently facing the research community in this field:

- *Design of defenses against new threats.* A new trend in the design of botnets is the use of existent, legal P2P networks, which are named *parasite botnets* [217]. This is the case of Storm [81], which uses the Overnet network to find and communicate with its bots. There exists also the example of Alureon/TDL4 [87], one of the most active botnets in 2010, which communicates through KAD, the distributed P2P network of eMule client. In view of their heightened importance, we suggest that parasite botnets should be very carefully considered and efficient detection system should be proposed.

- *Cross-stage detection of botnets.* Despite the great effort made to detect botnets, this problem remains unsolved. Our examination of the botnet life-cycle suggests that any effective mechanism focused on hindering a particular stage constitutes a valid defense against botnets. On the other hand, another point of view can also be found in our approach to describe botnets. As in cross-layer design paradigms recently proposed for other technologies, a "cross-stage" approach could be followed for the detection of botnets. As an example, mechanisms for the detection of bot spreading (recruitment stage) could be combined with approaches to detect marketing actions by botnet developers (marketing stage).

- *Defense against botnets at the marketing stage.* The majority of the contributions discussing the marketing stage focus on monetization and advertisement analysis. These works aim at giving an idea of the importance of the botnet problem. Thus, techniques should be developed to detect anomalous activity in a marketing environment: underground forums, social networks, etc. In a subsequent phase, these notifications should be correlated to direct the analysis toward attack vectors such as spam, DDoS and click fraud.

- *Response mechanisms against botnets.* The development of response mechanisms against the presence of botnets is also very scant. We believe that the main reason for this is the lack of efficient detection mechanisms, which is a prior condition for a response to be made.

- *Moving from size estimation to impact estimation.* Most botnet measurement studies concern the number of compromised bots within a particular botnet, *i.e.*, they are based on size estimation. As suggested in [219], the research focus should shift from measuring the size of botnets towards analyzing the impact of specific botnets of particular significance to society.

- *Exploring infection based on emerging technologies.* A common vector for the recruitment stage is based on exploiting software vulnerabilities. These vulnerabilities are particularly apparent in new software or technologies (for example, Android). The research community must focus on this issue in order to reduce the effects of new infection techniques.

A.4.4. Detection System of Parasite P2P Botnets (publications 2, 3 and 7))

The main intuition behind our proposal is the fact that resources shared by legitimate users in a P2P network (*legitimate resources*) will be accessed in a different way than resources shared by nodes belonging to a botnet (*botnet resources*). For this reason, we are interested in building models for both legitimate and botnet resources.

Then, we will suggest a detection architecture that relies on these models to detect botnet resources in P2P networks.

Our models are based on the evolution of the *number of P2P nodes that share a specific resource* over time. This way, let $n_r(k)$ be the number of nodes sharing a resource r during a period of time of duration δ which ends at $k \cdot \delta,\ k = 1, ..., K$.

Based on the behavior of $n_r(k)$ for legitimate popular P2P resources, our main hypothesis is that $n_r(k)$ will differ substantially from these patterns when r represents a botnet resource. This means that the detection of potential botnet resources would be possible by monitoring $n_r(k)$ and detecting deviations from the expected temporal sharing behavior.

The key problem to verify this hypothesis is that it is not possible to use experimental traces from botnet resources. To the best of our knowledge, nowadays there are no real active parasite botnets reported, possibly due to the fact that it is extremely difficult to detect them by existing methods. For this reason, our strategy is to build a theoretical model for the sharing evolution of botnet resources, assuming that botmasters must follow certain rules to maintain and operate their botnets. Here, it is important to consider two properties that could not be circumvented by botmasters without degrading botnets behavior or exposing them to active detection mechanisms:

1. **Botnet resources must be popular resources.** Botnets are composed of a huge number of bots or peers [136]. When botmasters update the bot code or send commands, all the bots must download a given botnet resource which contains those commands or updates. This implies that a large number of downloads will be observed for that botnet resource.

2. **Botnet resources must have a short life-time.** The period of time during which a botnet resource is being shared should be short, due to the fact that this file is directly pointing to all the members of the botnet, exposing them to known detection systems [238] [239].

These two rules imply that, during the first phase of a botnet resource sharing, a high level of popularity is expected and after that all the bots will stop sharing the botnet resource in a short period of time.

Resource-based Botnet Detection: Architecture and Operation

The architecture of our proposal for a resource-based botnet detection system is shown in Figure A.6. The data from the resource monitoring system is feeded into our system, where the three typical stages are considered:

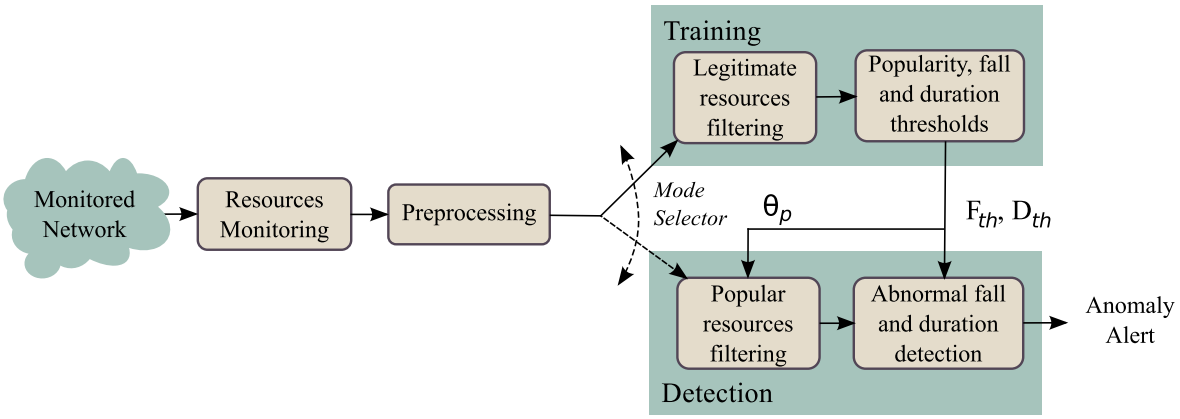


Figure A.6: Functional architecture of the detection system.

- *Preprocessing*

  Both the training and detection processes require a previous common stage to preprocess the data given by the resources monitoring system. We obtain $\boldsymbol{n_r}$, which represents a low pass filtering of the time series evolution of the number of peers sharing r.

- *Training*

  First, a normality model is built to represent the sharing evolution of legitimate resources in the monitored P2P network. As we are interested in identifying those resources that exhibit a high fall in the number of peers that share them and present a reduce sharing duration, we extract the fall threshold and the sharing duration of the system.

- *Detection*

  Once the model is obtained, every resource shared in the P2P network is analyzed, in order to determine potential deviations with respect to the expected behavior. In such a case, an alarm is triggered indicating the detection of a malicious, botnet resource. The core of the detection system corresponds to the module of *abnormal fall and duration detection*.

Experimental Evaluation

In order to make the training, we collect the evolution of 34,075 resources and we select 14,869 of them which are corroborated as legitimate. Additionally, 42,000 synthetic bot resources following our model based on popularity and short-life time are generated. The existence of both legitimate and botnet resources allows to obtain

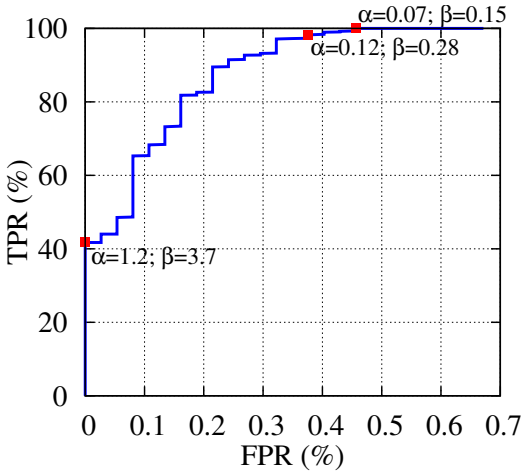


Figure A.7: ROC for our detection system by varying α and β.

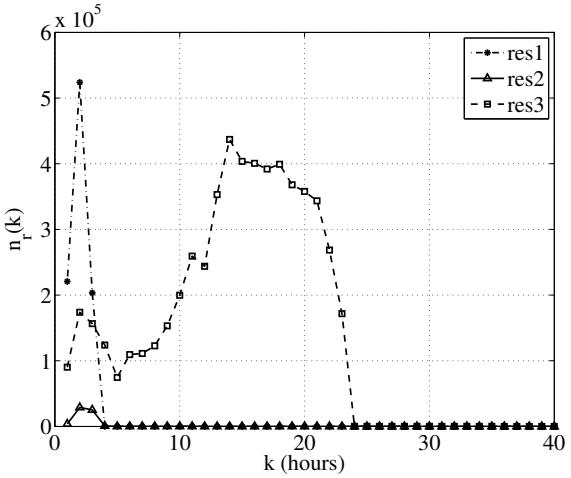


Figure A.8: Time evolution of the three resources detected as abnormal.

a ROC curve representing the performance of our detection system (see Figure A.7). We obtain this ROC by varying α and β which are the factor to calculate the fall threshold, F_{th}, and the duration threshold, D_{th}.

From these results, we select a value of α equal to 1.2 and a value of β equal to 3.7, which obtain FP rates equal to zero, in order to minimize the number of alarms while being sure about the malicious nature of the events detected.

Discovering Botnet Patterns

We have also carried out further experiments to check the system when new, unknown resources are observed and once the detector is completely tuned. For that, we used the remaining 19,206 resources obtained in the monitoring process but not previously used for training and testing our system. These resources could correspond either to legitimate or to botnet resources, as we only know that they are not explicitly recommended as valid by users. Our system raised alarms for only three of the monitored resources, which clearly behaved as outliers.

Figure A.8 shows $\boldsymbol{n_r}$ for these three resources. Here we can check that all of them are shared by a large amount of peers during a short period of time. Specifically, *res1* reaches a maximum of 523,883 users and the sharing phase of this resource lasts less than 5 hours. The same happens with *res2*, with less but still significative number of peers (28,897). Regarding *res3*, the duration of the sharing phase is longer, but still very short (only 24 hours), and the number of peers is really significative (436,963). These behaviors are really suspicious of being due to botnet resources sharing, as they follow the expected behavior regarding abrupt falling and short-duration in sharing.

Appendix B

Conclusions and Future Work

After describing the contributions of the work in Appendix A, here, we summarize the main conclusions of this thesis work. These have been indicated in each of the previous chapters, but here we present them in a synthetic and unified manner. Additionally, we describe the fundamental future works to be addressed in the line of research of this thesis.

B.1. Conclusions

Nowadays, the volume of traffic associated to P2P networks and more precisely to their applications, represents a considerable percentage of the global Internet traffic. However, the security in these networks is still an unsolved problem. A first step to provide security in these networks goes through the identification of the traffic that traverses the network. This way, for example, ISPs can establish certain policies to manage traffic in function of its type. Regarding P2P traffic classification we can highlight the following achievements in this thesis:

- We have started presenting a brief bibliographic review of the research in the traffic classification field. We have specially focused on P2P traffic classification.
- We have presented two network traffic classification systems: one specific for eDonkey protocol and the other one more general, which can be easily adapted to model and classify any type of network protocol.

- The proposed detection for eDonkey is based on the particular behavior of this protocol. We have designed two heuristic-based detection algorithms: the first one is intended to detect eDonkey flows (WCFD) and the second one to detect nodes that generate eDonkey traffic (URND).
- WCFD is based on the assumption that a peer that initiates a communication sends substantially more information than a peer that attends the communication. We conclude that this detection algorithm presents the limitation of being only able to detect flows for sharing resources between eDonkey peers with a high ID.
- URND is based on the hypothesis that peers that generate eDonkey traffic present a constant upload-rate and they communicate with multiple peers at the same time.
- Our proposal for a more generic detection system follows a methodology based on the use of *Hidden Markov Model*s (HMMs) to classify traffic. We contribute with a generic HMM at a flow level where observations are the packets of the analyzed flows. We particularize this generic HMM to classify eDonkey traffic as a case of study.
- We have carried out an experimentation with four network traffic traces, all of them composed of real network communications and two of them captured in a controlled environment. We conclude that both the detection algorithms for eDonkey traffic and the HMM-based scheme present a high true detection rate and a low false positives rate, they performing better than other proposals in the literature.
- Both classification systems are able to identify network traffic without inspecting packet payloads and, therefore, they respect privacy protection laws. Additionally, we have tested that they are capable of detecting obfuscated communications.

After P2P traffic identification, we have discussed that it is possible to apply security policies or anomaly detection/response systems over the identified network traffic to protect P2P networks against attacks to the security. One of the main issues nowadays in the field of security of P2P networks is that regarding botnets. Botnets constitute thus a topic of growing relevance, as we can notice in the number of research publications related with it. In this sense, the main conclusions of the present work are summarized in the following:

- We have proposed a generic life-cycle of botnets following a main idea: the stages that compose this life-cycle must be successfully executed to reach the final objective of botnets, *i.e.*, a successful attack. We have discussed that

hindering the execution of just one stage of the proposed life-cycle, the goal of the botnet will be thwarted.

- We have proposed a new taxonomy based on the previous cited life-cycle. This taxonomy is intended to group the research related with botnets, and thus to give some order to the huge amount of publications.

- We have reviewed the research works in the field of botnets following the proposed taxonomy, providing a global view of the most important contributions in this topic.

- We have contributed with a global perspective of the problem associated with botnets. This has allowed to reveal some deficiencies that should be taken into account by the research community in the next years. In this respect, we have indicated the most relevant research challenges to be addressed, together with some hints that can help to face them.

- We have suggested as a line of research to detect new threats in the field of botnets in an early state; precisely, to identify a specific type of botnet named P2P parasite botnet.

- We have presented a new approach to determine the occurrence of P2P parasite botnet events. The main novelty of this detection system is that it relies on monitoring the temporal evolution of resources shared by P2P networks instead on monitoring the network traffic itself. We claim that legitimate resources shared through P2P sharing networks present a time evolution that substantially differs from the evolution of botnet resources.

- We have used a methodology to recover the information patterns of resources shared in P2P networks. This methodology has been tested in the Mainline network, thus assuring its correct operation. The monitoring system used has allowed to define models of sharing patterns both for legitimate and botnet resources. We have developed a detection system based on the sharp falling and short duration of the sharing evolution of botnet resources.

- The experimental phase carried out has demonstrated that the proposed detection system exhibits great detection capabilities in terms of FPR (less than 0,4%) as well as in terms of TPR (more than 99%). Additionally, the simplicity of the proposed solution makes it appropriate for its implementation in real systems.

B.2. Future works

The main future works pointed out from the present thesis are summarized in the following:

1. To explore the possibility of identifying other P2P file sharing protocols (like BitTorrent) using the node detection algorithm URND. It can be assumed that the fundamental idea of this detection can be extended to other protocols because P2P file sharing protocols saturate the upload-rate bandwidth of users and this implies the necessity of a limit for it.

2. To extend the detection based on the limitation for upload-rate of P2P file sharing applications. We have noticed that these applications share the same constant upload-rate with each of the users they communicate with. Thus, P2P file sharing applications not only present a global quasi-constant upload-rate but a specific one per user. We try to develop a new detection algorithm based on this quasi-constant upload-rate with each of the users.

3. To carry out a deeper experimentation for the network traffic classification system based on HMM in order to probe its capabilities to represent and detect other protocols.

4. To use the traffic classification system based on HMM with the monitored information extracted from the sniffing module of Mainline. In the same way as the temporal evolution of shared resources of botnets present specific features, we believe that the sharing behavior of resources of different types (audio, video, software, etc.) will show certain particular features that can be represented by means of an HMM. This will be of a great importance in multiple fields, for example, for the detection of illegal sharing of resources protected by copyright.

5. To unify in a single detection system the algorithms URND and WCFD together with the network traffic model based on HMM. This way it will be possible, in a first step, to detect nodes that generate eDonkey traffic and, in a second step, to identify specific flows of these nodes that correspond to this (or other) protocol.

6. To completely automate the P2P parasite botnet detection system, integrating download and subsequent analysis of resources which raise alarms. We will be able to test if the detected resources are part of the communications of a parasite P2P botnet or not.

7. To increase the efficiency of the monitoring system over the Mainline network and thus be able to speed up the detection of P2P parasite botnet activities.

8. To develop a detection system for P2P parasite botnets with updates that occur only between a small subset of bots instead of among the complete botnet at the same time. The design of this new system should rely on the increase of the number of new resources generated and shared by the same subset of nodes in the network.