Breaking and Improving Protocol Obfuscation

Technical Report No. 2010-05, ISSN 1652-926X

Erik Hjelmvik
Independent Network Security
and Forensics Researcher
Enköping, Sweden
erik.hjelmvik@gmail.com

Wolfgang John
Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
wolfgang.john@chalmers.se

This document was prepared by Erik Hjelmvik and Wolfgang John for .SE (The Internet Infrastructure Foundation) and Chalmers University of Technology.

To be cited as:
Hjelmvik, E and John, W. *Breaking and Improving Protocol Obfuscation.* Department of Computer Science and Engineering, Chalmers University of Technology, Technical Report No. 2010-05, ISSN 1652-926X, 2010.

This technical report is available from:
`http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=123751`

Abstract

Different techniques for traffic classification are utilized in various fields of application. In this technical report, we look closer on how statistical analysis can be used to identify network protocols. We show how even obfuscated application layer protocols, such as BitTorrent's MSE protocol and Skype, can be identified by fingerprinting statistically measurable properties of TCP and UDP sessions. We also look closer on the properties our protocol identification algorithm exploits to identify these obfuscated protocols – protocols that are designed not to be detectable and are thus considered to be very hard to classify. Many of the analyzed protocols are shown to have statistically measurable properties in payload data, flow behavior, or both. Based on this new insight, we propose techniques that can improve future versions of obfuscated protocols, inhibiting identification through this type of statistical analysis. These techniques include better obfuscation of payload data and flow features as well as hiding inside tunnels of well known protocols. This report is intended to provide feedback and suggestions for improvement to creators of obfuscated network protocols, and should thus help to facilitate sustained network neutrality on the Internet.

Contents

1 Introduction **2**
1.1 Background . . . 2
1.2 Report Purpose . . . 3
1.3 Report Outline . . . 3

2 Related Work **4**
2.1 Classification by Payload Examination . . . 4
2.2 Classification by Social Host Behavior . . . 4
2.3 Classification by Statistical Flow Fingerprints . . . 5
2.4 Classification of Obfuscated Traffic . . . 5

3 The SPID Algorithm **6**
3.1 What is a Session? . . . 6
3.2 Overview of the SPID framework . . . 7
3.3 Attribute Meters . . . 7
3.4 Generation of Protocol Models . . . 7
3.5 Comparison of Protocol Models . . . 8

4 Training and Validation Data **9**

5 Breaking Protocol Obfuscation **11**
5.1 AttributeMeter Selection . . . 12
5.2 Threshold Adaptation . . . 13
5.3 Identification Results . . . 14

6 Analysis of Obfuscated Protocols **16**
6.1 Message Stream Encryption . . . 16
6.2 eDonkey TCP Protocol Obfuscation . . . 17
6.3 eDonkey UDP Protocol Obfuscation . . . 17
6.4 Skype TCP . . . 18
6.5 Skype UDP . . . 18
6.6 Spotify Server Protocol . . . 19
6.7 Spotify P2P Protocol . . . 20
6.8 Summary of Successful AttributeMeters . . . 21

7 Improving Protocol Obfuscation **23**
7.1 Obfuscation of Payload Data . . . 23
7.2 Obfuscation of Flow Features . . . 24
7.2.1 Randomized Flushing of Data Streams . . . 24
7.2.2 Random Padding . . . 24
7.2.3 Packet Directions . . . 24
7.3 Hiding Inside Well Known Protocols . . . 25

8 Conclusions **26**
8.1 Future Improvements of SPID . . . 26

Bibliography **29**

Chapter 1

Introduction

The concept of identifying protocols and applications through analysis of network traffic is known as "traffic classification". Techniques for traffic classification are used in many different applications, such as Quality of Service (QoS) assignments, traffic shaping, Intrusion Detection Systems (IDS) [Dreger et al., 2006] and in network forensics solutions. We also see a growing interest in Deep-Packet-Inspecting (DPI) firewalls that can block or allow network traffic based on protocol rather than on port numbers.

Traffic classification has also for some years been the favorite pet for researchers in the field of Internet measurements and analysis. Understanding which protocols and applications are being used on modern Internet, and tracking changes over time provides important knowledge of how the Internet needs to adapt in order to support future needs.

1.1 Background

The task of identifying application layer protocols in network traffic has during most of the Internet's lifetime been performed through simply looking at the TCP and UDP port numbers used by the server hosts[1]. The Internet Assigned Numbers Authority (IANA) has shouldered the burden of coordinating and managing a list of port-and-protocol mappings [Internet Assigned Numbers Authority, 2010] containing over 10,000 entries. Even though IANA state that *"Unassigned port numbers should not be used. The IANA will assign the number for the port after your application has been approved"* many new protocols are designed to use TCP and UDP ports for communication without respecting the authority of the IANA. The reasons for doing so can vary; many peer-to-peer (P2P) applications for example purposely avoid using known port numbers in order to dodge detection by traffic shapers. There are also other protocols, such as FTP data transfers and RTP[2], that use dynamically allocated ports for more legitimate reasons.

Multiple commercial and open source traffic classification solutions have been developed in order to thwart the technique of hiding the application and protocol through dynamic port allocation. The open source solutions, such as L7-filter [L7-filter, 2010] and OpenDPI [OpenDPI, 2010], mainly rely on pattern matching techniques. The concept of "pattern matching" involves identifying static byte sequences in the application layer data. The commercial implementations[3] are typically proprietary and don't provide much information on how they perform traffic classification, but a qualified guess is that they too rely on pattern matching. These traffic classification solutions are used by multiple ISPs in order to perform traffic shaping, i.e. blocking or limiting the bandwidth for, e.g., P2P file sharing applications[4] or Skype. The blocking of Skype is mostly carried out by telecom operators [O'Brien, 2010], who see Skype as a competitor to their traditional telephony services.

As a reaction to the use of traffic shaping several P2P applications started implementing protocol obfuscation. Techniques for achieving "protocol obfuscation" involves removing otherwise easily identifiable properties of protocols, such as deterministic byte sequences and packet sizes, by making the

[1] HTTP does for example often use TCP port 80 while DNS uses UDP port 53

[2] Real-time Transport Protocol (RTP) is used for transferring audio and video over the Internet (see RFC 3550)

[3] Ipoque, CISCO NBAR, SonicWall, Sandvine, Blue Coat PacketShaper and others

[4] A list of ISPs who block BitTorrent traffic is for example available at `http://wiki.vuze.com/w/Bad_ISPs`

data look as if it was random. The purpose is to make it impossible for the traffic classification engines to identify the application layer protocol. Common techniques for achieving protocol obfuscation is to apply encryption (to randomize the packet data) and include random sized paddings consisting of blobs with random data (to randomize packet sizes). Some examples of obfuscated protocols are Skype [Biondi and Desclaux, 2006], BitTorrent's Message Stream Encryption[5] and eMule's Protocol Obfuscation[6].

1.2 Report Purpose

The latest reaction in the arms race depicted above is that the traffic classification solutions are making attempts at also identifying obfuscated protocols. One purpose with the research presented in this report is to show that statistical methods can indeed be used to successfully identify even obfuscated protocols. The second, more important purpose of this report, is to shed light on the statistically measurable properties we have found in the obfuscated protocols which can be used in order to fingerprint these particular protocols. In other words, the purpose of this report is not to measure exactly how well the SPID algorithm can identify protocols, but rather to highlight the underlying statistical features that can be used to identify even obfuscated protocols. After all, as Sommer and Paxson puts it in [Sommer and Paxson, 2010], "*insight* matters much more than just numerical results". Even though Sommer and Paxson are referring to machine learning for intrusion detection we argue that the same reasoning also applies to traffic classification; presenting results only in means performance figures is not of much value unless *insight* can be gained on why these values are high or low. We also want to point out that our goal in revealing such insight is not to enable more effective filtering of P2P traffic on the Internet, but rather to provide feedback to protocol creators so that future protocol obfuscation implementations can be better engineered. We are thereby hoping to contribute to improved possibilities for network neutrality on the Internet.

1.3 Report Outline

In the following section (Chapter 2), we present related work on traffic classification research. In Chapter 3 we outline the SPID algorithm and give an overview of the protocol identification and classification framework we use in the present report. In Chapter 4 we introduce the datasets we use as training and validation data for SPID. Chapter 5 then describes how we optimized SPID for our chosen set of obfuscated protocols, and consequently presents the classification results of SPID on these protocols. In Chapter 6, we provide a detailed analysis of the obfuscated protocols we successfully classified, and highlight the attributes and features which proved to be most successful for each respective protocol. Chapter 7 offers some proposals on how protocol obfuscation can be improved based on the lessons learned from the in-depth protocol analysis in Chapter 6. Finally, Chapter 8 concludes this report and points to future directions for the SPID framework.

[5] `http://wiki.vuze.com/w/Message_Stream_Encryption`

[6] `http://wiki.emule-web.de/index.php/Protocol_obfuscation`

Chapter 2

Related Work

Traditionally, traffic classification was trivial since network applications have used fixed *port numbers* as publicly assigned by the IANA. However, with increasing popularity of P2P file sharing systems and their legal implications due to copyright concerns, popular applications started to use dynamic port ranges or reused ports assigned to other applications in order to disguise their activities. As a result, simple port classification performs rather poorly on modern network data [Moore and Papagiannaki, 2005, Madhukar and Williamson, 2006].

2.1 Classification by Payload Examination

The next obvious strategy for traffic classification was payload examination, also called *deep packet inspection (DPI)*. The simplest form of DPI is inspection of packet payloads for known, static, often manually created string patterns [Moore and Papagiannaki, 2005, Sen et al., 2004, Choi et al., 2004, Karagiannis et al., 2005, L7-filter, 2010, Dreger et al., 2006]. This type of classification is still considered one of the most reliable ways to classify traffic, and is thus also widely used in commercial tools such as Ipoque, Sandvine and others.

A problem with static signature matching is the high effort of keeping the signature updated. As a result, researchers proposed classification methods using automatically created application layer byte-level signatures [Haffner et al., 2005, Ma et al., 2006]. [Mantia et al., 2010] present Stochastic Packet Inspection (SPI) with the ability to identify application layer protocols running over TCP. In their research, they collected training data for protocols such as HTTP, FTP, Skype and SSH. They built protocol signatures consisting of entropy measurements of the first 24 nibbles in packets. As the authors themselves point out more robust solution could be achieved by statistically encode the expected application layer data (expected sequence) rather than the entropy measurements (expected frequency) of this data.

However, the ultimate drawback of DPI is the need for packet payload data, which raises privacy and legal concerns and as a result might not be accessible on some network locations.

2.2 Classification by Social Host Behavior

To mitigate the privacy related problems of DPI, various payload independent traffic classification methods have been proposed. One way is to classify the traffic based on the *social behavior of hosts* by looking at connection patterns of endpoints and trying to infer their application type by heuristic methods [Karagiannis et al., 2005, John and Tafvelin, 2008, Iliofotou et al., 2009, Dewaele et al., 2010, Trestian et al., 2010]. While some of these methods are quite successful, they only support classification of traffic into rough categories (e.g., WWW, mail or P2P traffic). Furthermore, these methods require sufficient traffic flows per endpoint in order for the heuristics to work, which is usually only given on links with high level of traffic aggregation (e.g. backbone links).

2.3 Classification by Statistical Flow Fingerprints

Another payload independent approach for traffic classification is the use of *statistical fingerprints* of flow properties. As an example, [Crotti et al., 2007] proposed a very basic approach of simple statistical fingerprinting by using three traffic features, i.e., packet sizes, packet inter-arrival times and packet arrival order. With this method, they were able to successfully classify the traditional protocols HTTP, SMTP and POP3.

The power of statistical methods for protocol identification has also been shown in various further papers proposing machine learning methods to classify traffic. Traditional protocols, such as HTTP, SMTP and FTP, have been classified in [McGregor et al., 2004] and [Erman et al., 2006] based on various flow properties, including flow duration, packets counts, inter-arrival times, etc. Other methods used similar flow properties to classify flows according to rough traffic categories (e.g., bulk, interactive, P2P, etc.) rather than specific protocols [Moore and Zuev, 2005, Roughan et al., 2004, Zuev and Moore, 2005].

Bernaille et al. showed that even the first four packets of a TCP session can be sufficient to classify many tradition protocols - both unencrypted (such as HTTP, SMPT, POP3); and encrypted (such as SSH, HTTPS, POP3S) [Bernaille et al., 2006]. In their work, only two features, namely means and variances of packet sizes and packet inter-arrival times, are applied within clustering based training techniques to perform early identification of network flows.

Complementary information about related work in the field of traffic classification can be found in the survey of traffic classification techniques using machine learning in [Nguyen and Armitage, 2008], in the comparison of contemporary classification methods in [Kim et al., 2008], the survey on Internet traffic identification in [Callado et al., 2009] and the research review on traffic classification in [Zhang et al., 2009].

2.4 Classification of Obfuscated Traffic

Classification efforts, focusing on obfuscated traffic, have so far mainly been carried out for individual protocols. Bonfiglio et al. successfully reveal Skype traffic, a both proprietary and encrypted, thus obfuscated VoIP protocol [Bonfiglio et al., 2007]. This is possible with a two-stage classifier: first, the statistical properties of the message content is used to fingerprint Skype's message framing; second, a Naive Bayesian techniques is used to identify statistical characteristics in flow features such as packet arrival rate and packet lengths.

Tunneling of application protocols inside other application layer protocols can also be considered as obfuscation, especially if the outer protocol is encrypted (e.g. HTTPS and SSH tunnels). Dusi et al. presented a mechanism called "Tunnel Hunter", which can successfully identify protocols tunneled inside tunneling protocols such as HTTP, DNS and SSH [Dusi et al., 2009]. This is done by statistical fingerprinting of simple IP level flow features, i.e. packets sizes, inter-arrival times and packet order.

A recent paper, [Bar-Yanai et al., 2010], presents a method for real-time classification of encrypted traffic. The proposed statistical classifier is based on a combination of k-means and k-nearest neighbor geometrical classifiers and is shown to be very robust even to obfuscated traffic such as Skype and encrypted BitTorrent. The statistical feature set is composed of 17 parameters based on packet and payload byte counts, packet sizes and packet rates for each direction (i.e., client to server, server to client).

Chapter 3

The SPID Algorithm

This chapter outlines the inner workings of the SPID algorithm, for a more complete description please see our paper"Statistical Protocol IDentification with SPID: Preliminary Results" [Hjelmvik and John, 2009] or download the SPID Proof-of-Concept source code from SourceForge[1]*.*

The main goal of the SPID algorithm is to reliably identify which application layer protocol is being used in a network communication session in a simple and efficient fashion. The SPID algorithm does not classify traffic into rough, coarse-grained traffic classes (such as P2P or web), but in fine-grained classes on per-protocol basis, which enables detailed QoS assignments as well as deep packet inspection of network traffic.

SPID is an acronym for "Statistical Protocol IDentification", and as the name suggests the SPID algorithm uses statistical measurements rather than relying on static pattern matching. Doing so eliminates the need for manually extracting inherent properties of protocols, since the SPID algorithm has the ability to automatically deduce properties from training data. This gives the SPID algorithm an advantage over pattern matching algorithms when it comes to identifying proprietary protocols[2] since manually creating signatures for proprietary protocols — such as Skype, Spotify's streaming protocol and botnet command-and-control (C&C) protocols — can be very troublesome. The training data used by the SPID algorithm does however need to be pre-classified, which can be done through manual classification by experts or with an atomized approach, eg. [Szabo et al., 2008, Gringoli et al., 2009].

Operational key requirements for the SPID algorithm are:

1. Small protocol database size
2. Low time complexity
3. Early identification of the protocol in a session
4. Reliable and accurate protocol identification

The motivation for requirements 1 and 2 are that it should be possible to run the SPID algorithm in real-time on an embedded network device, which has limited memory and processing capabilities. Requirement 3 should enable the use of the results from the SPID algorithm in a live traffic capturing environment in order to, in real-time, provide QoS to an active session, block illicit traffic or take the decision to store related traffic for off-line analysis. An implicit requirement is therefore that protocols should be identifiable based on the first few packets with application layer data.

3.1 What is a Session?

As of version 0.4.4 of the SPID Proof-of-Concept (PoC) both TCP sessions and UDP sessions can be identified. The concept of TCP sessions is well established; TCP sessions start with a three way handshake and end with a RST or FIN→FIN+ACK→ACK [RFC 793, 1981].

[1] `http://sourceforge.net/projects/spid`

[2] Protocols without publicly available specifications

For the notion of a "UDP session" we have in our research decided to define a UDP session as a set of UDP packets sharing the same 5-tuple[3], where packets going in both directions are part of the same session. We define the UDP session as being closed when 60 seconds have passed since a packet was last transmitted in the session. A packet with an identical 5-tuple arriving after such a 60 second timeout is thereby classified as being the first packet in a new UDP session. The client host of a UDP session is defined as the sender of the first packet in the session, the host receiving the first packet is thereby naturally classified as the server.

3.2 Overview of the SPID framework

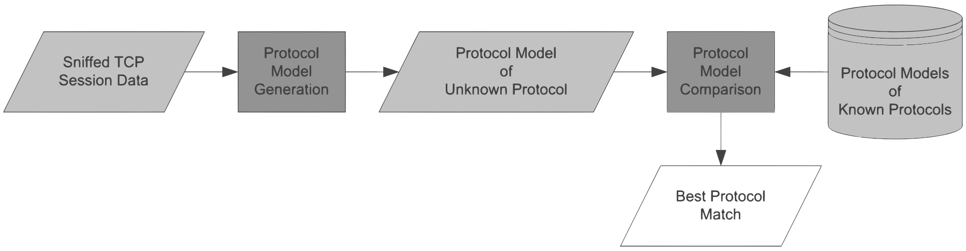
Sniffed TCP Session Data
Protocol Model Generation
Protocol Model of Unknown Protocol
Protocol Model Comparison
Protocol Models of Known Protocols
Best Protocol Match

Figure 3.1: Protocol identification data flow in SPID

The SPID algorithm performs protocol identification in a TCP or UDP session by comparing the Protocol Model of an observed session to pre-calculated Protocol Models of known protocols. Each such Protocol Model holds a set of attribute fingerprints, where each fingerprint is a probability distribution for the measured attribute (i.e. application layer data or flow feature). The fingerprints are created through performing frequency analysis of various attributes, such as application layer data values or flow features, and represented as probability distributions. Figure 3.1 illustrates the data flow of the SPID protocol identification. Our current implementation of the SPID algorithm performs individual classifications of each separate session, i.e. not keeping track of previously identified protocols for recurring IP/port pairs.

3.3 Attribute Meters

Each attribute fingerprint in a SPID Protocol Model is populated by measurements provided by an *AttributeMeter*. The PoC SPID application provides over 30 AttributeMeters [Hjelmvik, 2009] that are used to measure various generic behaviors of application layer protocols.

The attribute fingerprints are represented in the form of probability distributions. This means that the data for each fingerprint is represented by two arrays (vectors) of discrete bins; one array of counter bins and one array of probability bins. For each measurement provided by the AttributeMeter the value at the corresponding index in the counter vector is increased by one. The probability vectors are a normalized version of the counter vectors (values between 0.0 and 1.0), with all values in every probability vector summing up to 1.0. The data format used in this paper uses vectors of length 256; an implementation of the SPID algorithm can, however, use any length for these vectors. Longer vectors would give better precision, but would also require more training data and use more memory.

3.4 Generation of Protocol Models

For observed sessions a Protocol Model object is created upon session establishment (e.g. after the TCP three way handshake), consisting of a set of attribute fingerprints, as depicted in Figure 3.2. Every packet with application layer data belonging to a session is called an observation. Each such observation is then fed into the current session's Protocol Model object. Upon receiving an observation, the Protocol Model increments the fingerprint counters accordingly for each AttributeMeter.

[3] source IP, destination IP, source port, destination port, transport layer protocol (i.e. UDP)

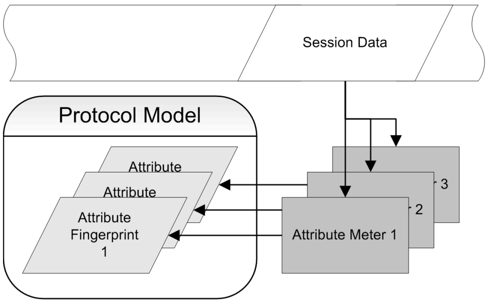

Figure 3.2: Generation of Protocol Models

All other attribute fingerprints, belonging to the same Protocol Model, will also increase their counters based on the sets of indexes (i.e. measurements) that are returned from their respective AttributeMeter. Subsequent packets in the same session will trigger the attribute fingerprint handlers of the session's Protocol Model to get more attribute measurements, which will cause the fingerprint counter vector values to further increment. However, one design goal of SPID was to keep time complexity low and to support protocol identification early during a session. For this reason, most current AttributeMeters do not consider more than the eight initial packets of a session, and typically less than 32 bytes in these packets (for details see Section 5.2 or [Hjelmvik, 2009]).

Protocol Models for known protocols are generated from real network packet traces. These traces need to be pre-classified, either manually or automatically, in order to be usable as training data for the SPID algorithm. The pre-classified training data is converted to Protocol Model objects (one per protocol) by generating Protocol Models for each session and merging the fingerprints of the same protocol and attribute type.

3.5 Comparison of Protocol Models

Fingerprints of an observed session are compared to fingerprints of known protocols (Protocol Models) by calculating the Kullback-Leibler [Kullback and Leibler, 1951] (K-L) divergence (also known as relative entropy) between the probability distributions of the observed session and each Protocol Model. For a given AttributeMeter, the best Protocol Model match for an observed session is the model which yields the smallest K-L divergence according to Eq. 3.1, where P_{attr} and $Q_{attr,prot}$ represent the probability vectors for a specific AttributeMeter ($attr$) of an observed session and of a known Protocol Model ($prot$) respectively.

$$D_{K-L}(P_{attr}||Q_{attr,prot}) = \sum_{i} P_{attr}(i) * log \frac{P_{attr}(i)}{Q_{attr,prot}(i)} \tag{3.1}$$

Protocol models of observed sessions are finally compared to Protocol Models of known protocols by calculating the K-L divergences of the models' attribute fingerprints. The best protocol match is the one with the smallest average K-L divergence of the underlying attribute fingerprints. A good approach is to assign a *K-L divergence threshold* value, where only K-L divergence average values under the threshold are considered as matches. If none of the known Protocol Models matches, the session is classified as 'Unknown' in order to avoid false-positives for known models.

Chapter 4

Training and Validation Data

Network traffic was collected from the applications Vuze, μTorrent, Skype, eMule and Spotify to be used as training and validation data. The protocols used by these applications carry properties such as being encrypted, proprietary or obfuscated. These three properties can be described as:

- **Encryption** — The primary purpose of encryption is to achieve confidentiality of encrypted data so that only those with knowledge of a special key can decrypt the data. A property of practically all encryption algorithms is that the encrypted data appears as random, i.e. the values are evenly distributed across the possible value space, regardless of what data is being encrypted.
- **Proprietary** — A proprietary protocol is a protocol for which there is no official and publicly available specification or source code. Proprietary protocols are typically designed and used by companies who do not want their protocols to be implemented or understood by other parties.
- **Obfuscation** — Protocol obfuscation is carried out with the purpose of making it difficult for a third party observer of a session to determine which application layer protocol is being used. Obfuscation is normally achieved by removing any otherwise measurable properties by making them appear random.
  - *Payload obfuscation* — Payload obfuscation has the goal to make it impossible for a third-party to identify a protocol based on observed byte patterns in the transport-layer payload. Typically, encryption is used in order to make the payload data of a protocol appear as random while still making it possible for the recipient to restore the original message.
  - *Flow-level obfuscation* — Flow-level obfuscation has the goal to obfuscate statistical flow properties, such as randomizing easily observable features like packet sizes and packet inter-arrival times.

The applications analyzed were all selected since they use protocols that are either obfuscated, proprietary, encrypted or all three of them, as shown in 4.1. The protocols used by Skype do for example

Application	**Version**	**Description**	**Url**	**Obfuscation**	**Proprietary**	**Encryption**
Vuze	4.3.0.6 - 2.4.0.0	BitTorrent client	`http://www.vuze.com/`	X		X
μTorrent	1.8.3	BitTorrent client	`http://www.utorrent.com/`	X		X
Skype	3.6.0	Microsoft Windows version	`http://www.skype.com/`	X	X	X
eMule	0.49c	eDonkey client	`http://www.emule-project.net/`	X		X
Spotify	0.3.17 - 0.3.23	Microsoft Windows version	`http://www.spotify.com/`		X	X

Table 4.1: Analyzed applications using obfuscated protocols

ISP	IP Network	Network Type	Bandwidth	Country
Bredbandsbolaget	85.226.216.0/21	Residential fiber	100Mbit/s	Sweden
TeliaSonera	81.224.0.0/12	ADSL	8Mbit/s	Sweden
Proxad	82.242.40.0/22	ADSL	2Mbit/s	France

Table 4.2: Traffic collection sites

employ encryption in order to achieve obfuscation and the protocol specifications are not publicly available (proprietary protocols). A couple of successful attempts at analyzing as well as reverse engineering Skype's protocol have, however, been performed [Bonfiglio et al., 2008, Biondi and Desclaux, 2006].

Our traffic was collected from three different domestic network connections in Sweden and France detailed in Table 4.2. The network traffic was collected by using Luigi Auriemma's Proxocket[1], which is a dll proxy for Winsock that dumps a copy of the network traffic to and from an application to a pcap file. The use of Proxocket enabled efficient separation of network traffic on a per-application basis. There is, however, not a one-to-one relationship between application and protocol. Network traffic from BitTorrent applications does for example often include not only the MSE protocol, but also normal unencrypted BitTorrent as well as HTTP[2] and the XBT Bittorrent Tracker protocol. We therefore created a Perl script called buildspiddb.pl[3] that performs this protocol filtering of the pcap files, this script is available under a GPL open source license at Google Code.

The training and validation data collected contained only the first 100 packets of each session, but with complete payload contents. We resticted data collection to the first 100 packets, since most current AttributeMeters have been designed to utilize only a limited number of packets and bytes per session [Hjelmvik, 2009] in order to enable early identification of protocols to support live analysis of traffic. Furthermore, the first few packets of a session are also usually the ones that reveal the most about what protocol is being used [Bernaille et al., 2006].

Just as in [Bar-Yanai et al., 2010] we noticed that our training and validation data contained a very large amount of short sessions, i.e. sessions with fewer than 15 payload packets. Instead of removing these sessions as in Bar-Yanai et al., we decided to keep them in both our training and validation data. We chose this approach because we want to be able to identify the application layer protocol of all sessions, which allows us to verify the robustness of SPID in the face of all types of traffic, including small flows.

[1] `http://aluigi.altervista.org/mytoolz.htm#proxocket`

[2] BitTorrent uses HTTP for torrent tracker announcements

[3] `http://code.google.com/p/buildspiddb/source/browse/trunk/buildspiddb.pl`

Chapter 5

Breaking Protocol Obfuscation

We evaluated the correctness and precision of SPID by aggregating the traffic collected from two of the locations (TeliaSonera and Proxad) into a training database. The traffic from the third location (Bredbandsbolaget) was used as validation data. By using different sites for training data and validation data we also evaluate the spatial stability [Pietrzyk et al., 2009] of the SPID algorithm.

The above described training data (see Chapter 4) was extended by Protocol Models for a garden variety of normal non-obfuscated protocols, namely BitTorrent, DNS, eDonkey, FTP, HTTP, IMAP, IRC, ISAKMP, MSN, POP, SMTP, SSH and SSL. The collection of training data for these protocols was performed as part of a earlier research related to the SPID algorithm [Hjelmvik and John, 2009]. The purpose of adding these protocols to the database was to make it harder for the SPID algorithm to select the correct protocol for a session by introducing a wider set of options to choose from.

Protocol	**Training Sessions**	**Validation Sessions**
BitTorrent	36	0
DNS	41	0
eDonkey	34	0
eDonkeyTCPObfuscation	1676	970
eDonkeyUDPObfuscation	875	1919
FTP	73	0
HTTP	121	0
IMAP	13	0
IRC	23	0
ISAKMP	22	0
MSE	1514	599
MSN	23	0
POP	26	0
SkypeTCP	585	98
SkypeUDP	3846	771
SMTP	27	0
SpotifyP2P	641	2653
SpotifyServer	20	30
SSH	54	0
SSL	81	0

Table 5.1: Number of Sessions in Training and Validation Data

Table 5.1 shows the number of sessions for each protocol collected as training data (i.e., to create Protocol Models) and validation data (i.e., to be classified with the Protocol Models created earlier).

5.1 AttributeMeter Selection

Based on related work and our expert knowledge, we initially designed 34 different AttributeMeters [Hjelmvik, 2009] for the SPID algorithm. Each such AttributeMeter provides individual measurements that can be used to identify application layer protocols. The strength of each individual AttributeMeter differs greatly, where some AttributeMeters perform fairly well even when used alone while other AttributeMeters reduces the overall precision of SPID when they are used. The amount of CPU power and memory required when running SPID is also proportional to the number of AttributeMeters used in parallel. We have therefore performed evaluations with the purpose of finding a reduced set of AttributeMeters that is most successful in differentiating between the seven obfuscated protocols in the validation data, when applying Protocol Models trained with the training sessions listed in Table 5.1.

While non-obfuscated protocols are typically easy to identify with (static) payload signatures, modern obfuscated protocols hide their content through encryption, leaving flow-level features as the most likely candidates for successful identification. We reason that a set of AttributeMeters that correctly identify and distinguish hard to classify obfuscated protocols should also work well when the task is to identify non-obfuscated protocols. However, it is likely that non-obfuscated protocols could be classified with even higher accuracy if AttributeMeters would be optimized for unencrypted payloads.

We decided to use a simple greedy algorithm to reduce the set of AttributeMeters. The selection algorithm was divided into a sequence of selection rounds, where one AttributeMeter was added to the set of previously selected AttributeMeters for each round. The logic for what AttributeMeter to select after each round is simple; the AttributeMeter that, in combination with the already selected set, correctly identifies most protocols is added to the selected set used in the next round. The greedy algorithm used Protocol Models that were built from the “Training Sessions” in Table 5.1 and used them to identify the “Validation Sessions” in the same table. An average True Positive (TP) ratio was calculated for each combination of validation protocol and AttributeMeter set. The average TP ratios for each AttributeMeter set were then averaged into a single TP percentage value, which was used to determine the AttributeMeter to select in each round. As shown in Figure 5.1, this TP percentage increases quickly during the first rounds, it then flattens out and finally starts to decrease after round 15.

The order of the first 15 AttributeMeters resulting of the greedy selection algorithm for the combined F-measure of the selected obfuscated protocols is shown in Table 5.2.

Rank	AttributeMeter	Packets/session	Type
1.	*FirstBitPositionsMeter*	8	payload
2.	*First2OrderedFirstBitPositionsMeter*	2	payload
3.	*AccumulatedDirectionBytesMeter*	4	flow-level
4.	*DirectionPacketLengthDistributionMeter*	100	flow-level
5.	*First4PacketsFirst32BytesEqualityMeter*	4	payload
6.	*First4PacketsByteFrequencyMeter*	4	payload
7.	*NibblePositionPopularityMeter*	8	payload
8.	*First2PacketsFirst8ByteHashDirectionCountsMeter*[1]	2	payload
9.	*ActionReactionFirst3ByteHashMeter*	5	payload
10.	*BytePairsReocurringCountIn32FirstBytesMeter*	100	payload
11.	*First4OrderedDirectionFirstNByteNibblesMeter*	4	payload
12.	*ByteValueOffsetHashOfFirst32BytesInFirst4PacketsMeter*	4	payload
13.	*ByteFrequencyOfFirstPacketBytesMeter*	8	payload
14.	*FirstServerPacketFirstBitPositionsMeter*	1	payload
15.	*ByteFrequencyMeter*[2]	5	payload

Table 5.2: Resulting order of greedy AttributeMeter selection. The table also shows the inspected number of packets per flow and the type of the AttributeMeter: based on *payload* or *flow-level* features.

The names of all AttributeMeter are to some extent self-explanatory, but a much more detailed description of each individual AttributeMeter can be found at the SPID Wiki [Hjelmvik, 2009].

[1] 92.715 percent of the sessions were correctly classified with 8 AttributeMeters

[2] 93.585 percent of the sessions were correctly classified with 15 AttributeMeters

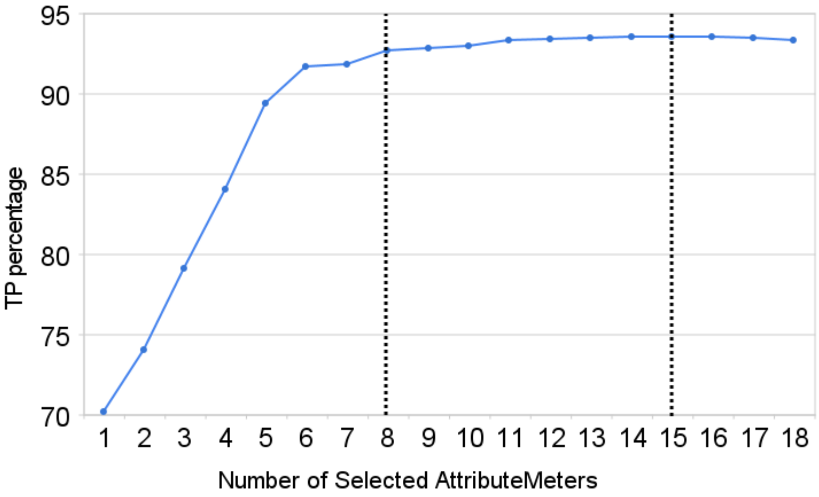

Figure 5.1: AttributeMeter selection rounds. Selected sets of 8 and 15 AttributeMeters are marked.

We decided to use two different sets of AttributeMeters; one small set of 8 AttributeMeters and one of 15. The small set is almost twice as fast as the larger one, which makes it more suitable when optimizing for speed. The larger set is on the other hand the combination of AttributeMeters that performed best in our evaluation, which therefore is better suited when more robust identification is needed.

5.2 Threshold Adaptation

The SPID algorithm uses a K-L divergence threshold in order to determine if the Protocol Model with the lowest K-L divergence (i.e. the best match) was not good enough and the session protocol therefore should be marked as 'Unknown'. During the AttributeMeter selection rounds a very high threshold value was used, thus yielding a forced choice of protocol for each session in the validation data. By adapting the threshold value so that it maximizes the average F-measure (see equation 5.3) for the sessions in the validation data we managed to find a suitable value for the threshold.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.3}$$

As can be seen in Figure 5.2, a low threshold generates a low recall rate. The reason for this is that most sessions are classified as 'Unknown', since not many Protocol Models yield K-L divergences below such a low threshold value. The precision is, however, high when using low threshold values, due to the fact that there are hardly any false positives. This means that the sessions that SPID attempts to identify with a low threshold value are mostly correctly classified. Figure 5.2 also show that the precision and recall converge at around 94 percent for high threshold values. The reason for the convergence is that the number of False Positives (FP) approaches the number of False Negatives (FN) when the threshold is high enough that SPID is forced to always choose a Protocol Model instead of marking the sessions as 'Unknown'.

The maximum average F-measure was achieved, for the 15 AttributeMeter set, when using a threshold value of 2.04 which generated an F-measure value of 94.1 percent. For the smaller 8 AttributeMeter set the best threshold value was at 1.94, yielding an F-measure of 93.1 percent.

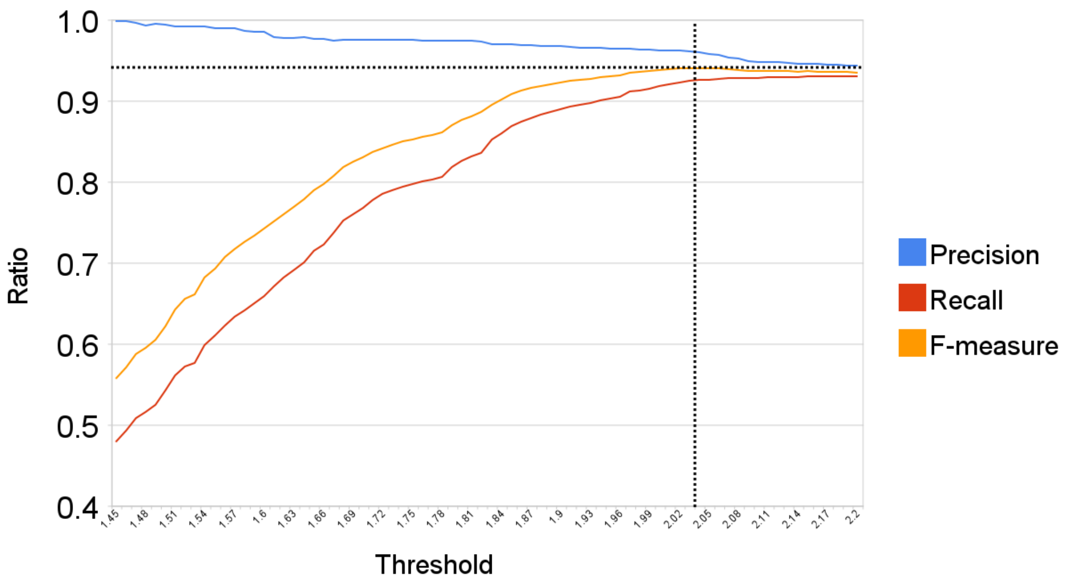

Figure 5.2: Threshold adaptation for 15 AttributeMeters. Selected threshold value (2.04) and the resulting F-measure (94.1 percent) are marked.

5.3 Identification Results

The classification performance of the SPID algorithm with the AttributeMeter set optimized for seven popular obfuscated protocols is best visualized by looking at a confusion matrix for the various protocols. Tables 5.3 and 5.4 show how the validation data for each of the obfuscated protocols was classified by SPID. Each row represents the real protocol and each column represents how it was classified by SPID. For presentation purposes, the 13 individual Protocol Models for the non-obfuscated protocols, included in the training data but not in the validation data (Table 5.1), have been summarized into the class 'Others'. The class 'Unknown' represents the sessions for which no Protocol Model yielded a K-L divergence below the threshold, thus SPID was not able to classify those sessions with sufficient confidence.

Note that some TCP protocols are even classified as UDP protocols and vice versa, e.g. eDonkeyTCPobfuscation is confused with eDonkeyUDPobfuscation in some cases. This is due to the current PoC implementation of SPID; it treats each session as an generic session, disregarding the transport protocol, which represents a worst case scenario for the traffic classification. We follow this approach because the goal in the current work is to identify effective statistical features rather than optimization of classification accuracy per se. However, it would be trivial to take transport protocols into account in a pre-filtering step before applying protocols models on the collected data, which would further increase the accuracy of SPID. We therefore plan to include this functionality for future releases of SPID.

In general, the results in Tables 5.3 and 5.4 show that SPID could correctly classify more than 90 percent of the sessions for all protocols except SkypeTCP and SkypeUDP. The Skype traffic was what SPID had most trouble identifying, where only 39 percent of the SkypeTCP sessions and 66 percent of the SkypeUDP sessions were correctly classified when using 15 AttributeMeters. The results for Skype was, however, significantly better when only using 8 AttributeMeters.

One of the reasons for why we got such poor results for Skype can be due to the fact that Skype uses different codecs [Bonfiglio et al., 2008], where we believe the codec is selected based on available bandwidth and latency. In our research the training data (from TeliaSonera and Proxad) was captured on ADSL networks, while the validation data was captured on a 100 Mbit/s residential fiber network (Bredbandsbolaget). This could cause the Skype training data to differ more than expected from the validation data. We have in additional evaluations (not shown here) validated that the results for Skype become much better when choosing the training and validation data from location with comparable access bandwidths.

However, as pointed out earlier in the introduction, the purpose of this report is not to measure just how well the SPID algorithm can identify protocols, but rather to show how statistical methods can be used to identify even obfuscated protocols. For this reason, we do not provide any overall values for recall, precision and F-measure, but rather focus on gaining *insight*, which will be discussed in detail in the following Chapter 6.

Protocol	MSE (encr. BT)	SkypeTCP	SkypeUDP	SpotifyP2P	SpotifyServer	eDonkeyTCPObfuscation	eDonkeyUDPObfuscation	Other (13 protocols)	Unknown
MSE (encr. BT)	**0.963**	0	0	0	0	0.002	0.003	0	0.032
SkypeTCP	0.051	**0.653**	0	0	0.010	0.031	0.194	0.020	0.041
SkypeUDP	0	0.001	**0.767**	0	0	0	0.001	0.001	0.230
SpotifyP2P	0	0	0	**0.913**	0.039	0	0	0	0.049
SpotifyServer	0	0	0	0	**0.933**	0	0	0	0.067
eDonkeyTCPObfuscation	0.047	0	0	0	0.002	**0.910**	0.005	0.009	0.026
eDonkeyUDPObfuscation	0.001	0.002	0.001	0	0	0.001	**0.973**	0.001	0.021

Table 5.3: Confusion matrix for SPID classifications of obfuscated protocols with 8 AttributeMeters and a K-L divergence threshold of 1.94. The values in each row present the percentage of sessions classified as the respective protocols in columns.

Protocol	MSE (encr. BT)	SkypeTCP	SkypeUDP	SpotifyP2P	SpotifyServer	eDonkeyTCPObfuscation	eDonkeyUDPObfuscation	Other (13 protocols)	Unknown
MSE (encr. BT)	**0.965**	0	0	0	0	0	0.005	0	0.030
SkypeTCP	0.051	**0.388**	0	0	0	0.041	0.500	0	0.020
SkypeUDP	0.001	0	**0.657**	0	0.001	0	0.001	0	0.340
SpotifyP2P	0	0	0	**0.932**	0	0	0	0	0.068
SpotifyServer	0	0	0	0	**0.967**	0	0	0	0.033
eDonkeyTCPObfuscation	0.030	0	0	0	0	**0.929**	0.006	0	0.035
eDonkeyUDPObfuscation	0.001	0	0.001	0	0	0.001	**0.974**	0	0.023

Table 5.4: Confusion matrix for SPID classifications of obfuscated protocols with 15 AttributeMeters and a K-L divergence threshold of 2.04. The values in each row present the percentage of sessions classified as the respective protocols in columns.

Chapter 6

Analysis of Obfuscated Protocols

The purpose of our research is not only to show how statistical methods can be used to identify obfuscated application layer protocols, but more importantly to provide feedback to the networking community (including designers of obfuscated protocols) so that they can gain insight on how protocol obfuscation can be improved. In this chapter, we therefore point out the statistically measurable behavior of each protocol that the SPID algorithm was able to use in order perform successful protocol identification.

For each of the seven protocols analyzed, we list those AttributeMeters that turned out to be most successful for the respective protocol. The AttributeMeters considered here are all 34 AttributeMeters we initially developed [Hjelmvik, 2009]. Note that this approach differs from the AttributeMeter selection described in Section 5.1, since now we search for the optimal AttributeMeters for *each individual protcol*, as opposed to the search of an optimal AttributeMeter set for *seven protocols combined* as done earlier.

6.1 Message Stream Encryption

Message Stream Encryption (MSE) is the obfuscation protocol used for BitTorrent implementations. The MSE protocol is also known as Protocol Header Encryption (PHE).

The MSE specification [Vuze Wiki, 2010] defines that the client, after a completed TCP three-way handshake, shall send a 96 byte Diffie-Hellman [Diffie and Hellman, 1976] public key ($G^{Xa} \bmod P$) and a padding of 0 to 512 bytes to the server. The size of this data will therefore always be in the range between 96 and 608 bytes. This initial data chunk can also be segmented into multiple TCP packets if the client wishes to do so. The server then responds with its own Diffie-Hellman public key ($G^{Xb} \bmod P$) plus padding, which also will be in the range of 96 to 608 bytes and can be segmented into several TCP packets. The client then completes the Diffie-Hellman transaction and chooses an encryption algorithm for the rest of the TCP session by sending a chunk of data in the range of 124 to 636 bytes. These three initial exchanges between the client and server are performed as blocking steps [Vuze Wiki, 2010], meaning that each step need to be finished before the next one starts.

Vuze, which is one of the two BitTorrent applications from which we have captured MSE traffic, is available as open source at SourceForge[1]. Analysis of `ProtocolDecoderPHE.java` in the Vuze source code reveals that it does not implement the full randomness provided by the MSE specification. The padding in Vuze is only half the size of what is specified in the MSE specification, i.e. between 0 and 256 bytes. This yields a maximum size of 352 bytes for the first two chunks of data being sent from client to server and vice versa.

Analysis of network traffic from both Vuze and μTorrent show that the padding seem to be limited to maximum 256 bytes in both applications. The reduction of the maximum padding size from 512 to 256 bytes was most likely performed in order to reduce the traffic overhead of the MSE protocol. However, we believe this reduction negatively affects the protocol's ability to avoid detection from the SPID algorithm. Our traffic analysis also shows that the segmenting of the initial data into multiple packets often yields TCP packets with around 100 bytes of application layer data. Our analysis even shows that the first packet in more than half of the sessions has between 64 and 128 bytes of application layer data.

[1] `http://sourceforge.net/projects/azureus`

Of our 34 AttributeMeters the ones that best capture these properties of MSE are:

- *AccumulatedDirectionBytesMeter*
- *DirectionPacketLengthDistributionMeter*

Both these AttributeMeters measure packet sizes and packet directions, but with slightly different approaches. We therefore assume that flow-level analysis would be more effective than payload analysis when it comes to identifying MSE traffic. It seems, however, to be of advantage to include both payload and flow-level analysis as done with SPID.

6.2 eDonkey TCP Protocol Obfuscation

Analysis of the source code from eMule (the de-facto eDonkey client) provides some interesting details about the protocol obfuscation provided by eMule for TCP[2]. The first four packets of each session will behave as follows:

Packet 1. Client→Server: 12 to 267 bytes

Packet 2. Server→Client: 6 to 261 bytes

Packet 3. Client→Server: 98 to 113 bytes

Packet 4. Server→Client: 103 to 358 bytes

Statistical analysis of the protocol also shows that, out of the first 20 packets in a session, the most common packet sizes from client to server are 9 to 11 bytes (22.6 percent) and 54 to 59 bytes (22.7 percent). After completed encryption handshake the packet sizes lie at the Maximum Segment Size (MSS).

The SPID AttributeMeters that best capture these properties of the eDonkey TCP protocol obfuscation protocol are:

- *AccumulatedDirectionBytesMeter*
- *DirectionPacketLengthDistributionMeter*
- *First4OrderedDirectionPacketSizeMeter*

These three AttributeMeters do all measure flow-level properties, such as packet directions and sizes. Thus flow analysis is most likely the most effective way of identifying eDonkey TCP Protocol Obfuscation.

6.3 eDonkey UDP Protocol Obfuscation

Analysis of our collected traffic for the eDonkey UDP protocol obfuscation, also known as Encrypted-DatagramSocket, show that the first four packets in a session very often look as follows:

Packet 1. Client→Server: 51 bytes

Packet 2. Server→Client: 85 bytes

Packet 3. Client→Server: 44 bytes

Packet 4. Server→Client: over 900 bytes

Our analysis also shows that the least significant bit in the first byte (i.e. bit 8) practically always (97 percent of the sessions) has the value 0. This means that the first byte can be expected to have an even value for eDonkey UDP protocol obfuscation.

Another interesting property of this protocol is that packets with application layer data sizes divisible by two are less common than those divisible by three. For truly randomly distributed packet sizes, on the other hand, there would be 50 percent more packets whose sizes are divisible by two compared to those divisible by three.

[2] The obfuscation protocol for TCP is named EncryptedStreamSocket in the eDonkey source code

The AttributeMeters which best capture these properties are:

- *FirstBitPositionsMeter*
- *PacketPairLengthPrimesMeter*
- *PacketLengthDistributionMeter*

The *FirstBitPositionsMeter* measures the positions and values of the first 128 bits (i.e. 16 bytes) in the first 8 packets in each session, which means it measures payload properties. The other two AttributeMeters again measure flow-level features, i.e. the packet sizes of packets in a session.

6.4 Skype TCP

Being a truely proprietary protocol, there is neither any official specification nor any open source implementation available for Skype protocols and clients. Our analysis of the Skype protocols is therefore limited to statistical analysis of the captured Skype traffic. Due to the use of RC4 encryption in the Skype TCP protocol [Biondi and Desclaux, 2006] we are even further limited to consider mainly flow features.

Normal VoIP traffic has fixed packet lengths, but the codec used by Skype generates packet lengths between 20 and 200 bytes [Do and Branch, 2009]. Our statistical analysis of Skype TCP reveals the following packet size properties of the first four TCP packets with payload in a session:

Packet 1. Client→Server: 97 percent of the sessions send between 0 and 127 bytes

Packet 2. Server→Client: 94 percent of the sessions then send between 0 and 127 bytes back to the client

Packet 3. Client→Server: 99 percent of the sessions then send between 0 and 63 bytes

Packet 4. Server→Client: 100 percent of the analyzed sessions then send between 0 and 63 bytes back to the client

We observed also further noticeable properties of the Skype TCP protocol:

- The first 2 packets actually contain less than 69 bytes of application layer data in more than 90 percent of the sessions, they also rarely contain fewer than 39 bytes. Thus, the first two packets in the Skype TCP protocol are often between 39 and 69 bytes long.
- Bit values of 1 seem to be more common than bit values of 0.

The AttributeMeters in SPID that makes use of these properties are:

- *AccumulatedDirectionBytesMeter*
- *ByteBitValueMeter*
- *First2PacketsFirst3ByteHashAndPacketLengthMeter*

The first AttributeMeter here (*AccumulatedDirectionBytesMeter*) only measure flow-level properties, while the second one (*ByteBitValueMeter*) only measures payload properties. The last AttributeMeter (*First2PacketsFirst3ByteHashAndPacketLengthMeter*) combines flow and payload properties into its measurements. We therefore believe that a combination of flow and payload measurements would be the most effective method for identifying SkypeTCP traffic.

6.5 Skype UDP

The preferred transport protocol for Skype is UDP, which is used in more than 68 percent [Bonfiglio et al., 2008] of Skype's traffic. Just as for Skype's TCP protocol the UDP version also employs RC4 encryption. However, applying encryption to UDP based protocols is not as straight forward as for TCP based protocols since UDP lacks properties such as ordered and reliable delivery of data. For this reason, UDP protocols typically need to include some sort of header in each packet that defines an initialization vector, encryption key or something that can be used as a seed for encryption.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Frame ID number         | Function code | Encr. payload |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Encrypted payload (contd.)...                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6.1: Packet structure of the Skype UDP protocol

The first two bytes of each application layer packet are according to [Biondi and Desclaux, 2006] an ID number for the frame. We have in our analysis noticed that this ID number is not always fully random; many hosts seem to increment the frame ID number by some small random value rather than generating a completely new 16-bit value. We have in our analysis also found that the function code, which follows the frame ID field (see Figure 6.1), always has a value of $0x02$ in the first packets of each session. After the function code follows the payload, which has been encrypted with RC4. The RC4 encryption is there only with the purpose of providing a layer of obfuscation, not in order to provide confidentiality. There is in fact no secret key protecting the RC4 encryption, only a proprietary key expansion algorithm that was recently reverse engineered [O'Neil, 2010]. This implies that a DPI solution could in fact decrypt and inspect the RC4 encrypted payload of Skype for classification purposes. However, such processing would impose additional processing requirements and is therefore unfeasible for many network scenarios. One slight benefit that Skype gets from not using a Diffie-Hellman key exchange is that there is no initial key exchange in the beginning of each session that needs to be hidden from traffic classification engines.

The AttributeMeters in SPID that capture these properties are:

- $ByteFrequencyOfFirstPacketBytesMeter$
- $First2OrderedFirstBitPositionsMeter$
- $FirstBitPositionsMeter$
- $FirstServerPacketFirstBitPositionsMeter$
- $NibblePositionFrequencyMeter$
- $NibblePositionPopularityMeter$

All of these AttributeMeters use payload data to build their measurements. This means that the Skype UDP protocol is much better detected when using payload properties compared to when using flow properties, which we believe to be a direct result from the function code not being encrypted or obfuscated in the start of each UDP session. We are in fact surprised that the designers of the Skype UDP protocol chose not to encrypt or obfuscate the function code, since this single byte of "leaked" information makes it much easier to identify the Skype UDP protocol.

Other researchers have identified statistical properties in the packet sizes of Skype packets, where the packet sizes are claimed to be almost constant sized [Bonfiglio et al., 2008] or vary between 20 bytes and 200 bytes [Do and Branch, 2009]. These differing observations could be explained by the fact that Skype behaves differently depending on which codec it decides to use for a session [Bonfiglio et al., 2008].

6.6 Spotify Server Protocol

Spotify uses three different protocols for communication; SSL, Spotify Server Protocol and Spotify P2P Protocol. SSL is used for all GUI related communication, while all music that is streamed from Spotify's servers is delivered with the Spotify Server Protocol.

The Spotify Server protocol is clearly not designed to be obfuscated or in any other way avoid detection. In fact it can easily be identified even with simple static flow properties such as IP addresses and port numbers since the Spotify streaming servers reside on the 78.31.8.0/22 network and host the streaming service on TCP port 4070. The Spotify protocols are, however, proprietary. This means that there is no official protocol specification available for them. The protocols also incorporate encryption

Offset	#Bytes	Description
0	2	Protocol version (current version number is 3)
2	2	Packet length
4	1	Client OS (Windows = $0x00$, Mac = $0x01$)
5	4	Client ID
9	4	Client Revision
13	16	Random number from Client
29	96	Client public encryption key (for Shannon cipher)
253	8	Client username length (L_u)
254	L_u	Client username

Table 6.1: Contents of the first packet sent by the client in the Spotify Server protocol according to the despotify source code [despotify, 2010]

in order to prevent ripping of the streamed music. Because of this we still find it interesting to see how well SPID can be used to identify the protocols used by Spotify.

The Spotify implementation has been reverse engineered by a group of Swedish hackers who wish to remain anonymous. This group has released their own Spotify implementation as open source under the name "despotify" [despotify, 2010]. Analysis of the source code for despotify gives a good view of some properties in the Spotify Server protocol, even though the purpose of several fields in the protocol still are unknown to the despotify team.

The contents of the first packet in a Spotify Server Protocol session has a very distinct structure as described in Table 6.1. Some additional properties in the Spotify Server protocol that are worth mentioning are:

- The first packet with application layer data in the TCP session is sent from client to server
- The first two bytes of the first packet contain the protocol version (0x0003 for the current version of Spotify)
- The next four bytes contain a field that represents the length of the packet
- The last few bytes of the first packet contain the username associated with the Spotify user account

The AttributeMeters in SPID that capture these properties are:

- $ByteFrequencyOfFirstPacketBytesMeter$
- $First2OrderedFirstBitPositionsMeter$

Both these AttributeMeters measure properties in the payload data of the protocol.

6.7 Spotify P2P Protocol

Spotify also has a peer-to-peer (P2P) protocol [Kreitz, 2009], which is designed to offload the Spotify streaming servers. The Spotify P2P protocol is not designed with purpose of providing obfuscation, but it is still of interest in our research since it is a proprietary protocol that uses encryption. Thus, it is difficult to identify the protocol with traditional techniques even though it isn't purposely obfuscated.

Our statistical analysis of the Spotify P2P protocol shows some obvious properties:

- The first packet with application layer data in each session goes from the client to the server
- The first byte in the first chunk of data in each direction starts with the value $0x00$ (some exceptions have been observed where the first byte is $0x01$)
- TCP packets exchanged in the start of each session do typically have between 28 and 53 bytes of payload in both directions
- TCP packets later on in the session do usually go from server to client and have packet sizes limited by the MTU[3].

[3] Maximum Transmission Unit of an IP packet's payload, typically around 1500 bytes

The AttributeMeters in SPID that capture these properties are:

- *First2OrderedFirstBitPositionsMeter*
- *ByteFrequencyOfFirstPacketBytesMeter*
- *ByteFrequencyMeter*
- *First4PacketsFirst32BytesEqualityMeter*
- *First2PacketsFirst8ByteHashDirectionCountsMeter*
- *NibblePositionFrequencyMeter*
- *First4PacketsByteFrequencyMeter*
- *DirectionPacketLengthDistributionMeter*
- *FirstBitPositionsMeter*

All these AttributeMeters except *DirectionPacketLengthDistributionMeter* measure payload properties. We can thereby deduce that there is enough unencrypted payload data in the Spotify P2P protocol to support robust protocol identification based on payload analysis.

6.8 Summary of Successful AttributeMeters

Table 6.2 summarizes the successful AttributeMeters according to our analysis of each individual obfuscated protocol considered. It is important to note that this table does not include exactly the same AttributeMeters as listed in Table 5.2 (Section 5.1), where the set of AttributeMeters was optimized for a combined performance over all obfuscated protocols. To allow easier comparison, we included the ranks of the AttributeMeters according to the greedy selection algorithm also in Table 6.2 (last column).

Not surprisingly, the first eight AttributeMeters from Table 5.2 are also effective for identification of the individual protocols. However, four AttributeMeters that increased the combined overall performance according to the greedy selection process did not perform outstanding for any individual protocol, but contribute merely to classification accuracy in combination with other AttributeMeters. On the other hand, six AttributeMeters, that did not belong to the top fifteen combined AttributeMeters, performed well on an individual per-protocol basis. This can be explained by the fact that some AttributeMeters may contribute significantly to the number of true positive identified sessions of one protocol, while at the same time increasing the number of false positives for other protocols. As a result, such an AttributeMeter introduces a negative effect on the combined performance, measured over the success on all protocols.

The table shows that the protocols without careful payload obfuscation (i.e. the Spotify protocols and the Skype UDP protocol with the deterministic ID field) were best detected by using payload based measurements. The well obfuscated protocols have on the other hand more or less eradicated payload properties by using encryption. Specifically we note that the *DirectionPacketLengthDistributionMeter* and the *AccumulatedDirectionBytesMeter* seem to be fairly effective at identifying statistical properties in the obfuscated protocols. The first named AttributeMeter combines measures for packet sizes and packet directions. However, this AttributeMeter could be easily circumvented by proper payload padding (see Section 7.2.2). The latter named AttributeMeter therefore measures the consecutive number of bytes sent in each direction before the data starts flowing in the opposite direction. Since data is accumulated until a directional change, this AttributeMeter is more robust to randomized flushing (Section 7.2.1). An effective way of circumventing also this AttributeMeter is, in addition to padding, to eliminate predictable sequences of data exchanged between client and server.

	MSE (encr. BT)	eDonkeyTCPObfuscation	eDonkeyUDPObfuscation	SkypeTCP	SkypeUDP	SpotifyServer	SpotifyP2P	Rank (acc. to table 5.2)
Flow Meters								
AccumulatedDirectionBytesMeter	X	X		X				3.
DirectionPacketLengthDistributionMeter	X	X					X	4.
First4OrderedDirectionPacketSizeMeter		X						
PacketLengthDistributionMeter			X					
PacketPairLengthPrimesMeter			X					
Hybrid Meters								
First2PacketsFirst3ByteHashAndPacketLengthMeter				X				
Payload Meters								
ByteBitValueMeter				X				
ByteFrequencyMeter							X	15.
ByteFrequencyOfFirstPacketBytesMeter					X	X	X	13.
First2OrderedFirstBitPositionsMeter					X	X	X	2.
First2PacketsFirst8ByteHashDirectionCountsMeter							X	8.
First4PacketsByteFrequencyMeter							X	6.
First4PacketsFirst32BytesEqualityMeter							X	5.
FirstBitPositionsMeter			X		X		X	1.
FirstServerPacketFirstBitPositionsMeter					X			14.
NibblePositionFrequencyMeter					X		X	
NibblePositionPopularityMeter					X			7.

Table 6.2: Overview of the most successfull AttributeMeters for each individual protocol. For comparision, also the rank of the greedy AttributeMeter selection for combined protocol identification is listed.

Chapter 7

Improving Protocol Obfuscation

The purpose with our research is not to reinforce active filtering of P2P traffic on the Internet. Instead we want to support the concept of network neutrality by providing feedback to the creators of obfuscated protocols. As we have observed, the supposed-to-be obfuscated protocols are not obfuscated enough to avoid statistical identification of various properties specific to the protocols. Thus, there is a need to improve the design of these protocols in order to provide obfuscation that effectively can avoid traffic shaping techniques that employ deep packet inspection and statistical analysis. Some considerations of how to increase obfuscation by making data patterns to appear more random are discussed in the following sections.

7.1 Obfuscation of Payload Data

Payload data values in the protocols need to be randomly distributed, which can easily be achieved by applying encryption. Even light weight cryptos, such as RC4 [Rivest, 1992], are sufficient in order to provide a good random distribution of the data.

An obstacle to overcome when using cryptography is, however, the need for the client and server to exchange information in clear text prior to establishing the encrypted session. Information that typically need to be exchanged before applying encryption is application version numbers, supported crypto algorithms and — most importantly — the client and server need to agree on a session key to be used for the symmetric encryption. Each packet in a UDP based protocol additionally need to contain some field to be used as input to the encryption algorithm in order to protect the contents of that particular packet.

The most common technique for the client and server to establish a shared session key is to use Diffie-Hellman (D-H) key exchange [Diffie and Hellman, 1976]. D-H implementations in protocols such as SSH and SSL are very easily identified due to the overhead data used in the key exchange. As described in this report, the MSE protocol uses a much more compact, and thereby more random looking representation of the D-H key exchange data. In this D-H key exchange only the public key (which is random looking) is sent without any prepended packet header. The MSE D-H key exchange can, however, still be fingerprinted with statistical algorithms due to identifiable flow features such as deterministic packet sizes and directions used in the start of each MSE session. We therefore believe that the payload obfuscation provided by the MSE D-H key exchange is adequate as long as it is combined with proper flow obfuscation.

One possible alternative that can be considered when it comes to establishing a session key for encryption is to use a shared secret instead of performing a D-H key exchange. The shared secret could be a pre-shared key of some sort, but it could just as well be some sort of information that both the server and the client could calculate individually for each session. Skype does for example combine the server IP, client IP, the ID field and an initialization vector in order to produce a seed for its RC4 key expansion algorithm [Biondi and Desclaux, 2006]. Another option that would work for BitTorrent applications would be to use the torrent hash or some other public property of the file to be transmitted as a shared secret. Using this type of shared secret does not provide as strong protection as when performing a D-H key exchange, since an eavesdropper would be able to decrypt the communication if she too can get hold of the shared secret. On the other hand, this type of weak encryption might still be sufficient to avoid detection by DPI methods since it introduces a significant computational overhead. However, we suggest using D-H when possible.

7.2 Obfuscation of Flow Features

There are many flow properties that can be measured with statistical analysis, and therefore need to be removed from a protocol in order to achieve a good level of obfuscation. These properties include for example packet sizes and inter-arrival times.

7.2.1 Randomized Flushing of Data Streams

A simple way to achieve more random packet sizes for TCP based application layer protocols is to flush the TCP stream at random offsets into the stream. Performing the flush operation causes the TCP socket to transmit the data queued for sending instead of buffering the data to fill up the full Maximum Transmission Unit (MTU) of the underlying network layer protocol (typically IPv4). A good property of randomized flushing is that it is an obfuscation feature that doesn't affect the application layer protocol. This means that an application can apply randomized flushing while still being able to communicate with older versions of the application that do not yet support randomized flushing[1].

The flush operation cannot be used by UDP based protocols since they need to transfer their application layer data in datagrams instead of in a stream. The flush operation is also not of much use in TCP based protocols if only a few bytes can be sent before a response is required from the remote host.

7.2.2 Random Padding

A good approach to improve the randomness of the packet sizes further is to insert padding of random sizes into the transmitted data. The range of possible sizes of this padding need to be large enough in order to provide sufficient obfuscation. The MSE implementations in Vuze and μTorrent that limited the padding to maximum 256 bytes did not provide for enough variance. We therefore suggest that the padding need to be allowed to grow up to at least the Maximum Segment Size (MSS), which normally is around 1500 bytes.

A simple way to represent the random padding is to use one or two bytes to represent the length (in bytes) of the padding, followed by the specified number of random bytes. The use of padding does of course reduce the bandwidth efficiency of the protocol, so excessive use of random padding shall be avoided when throughput is critical.

7.2.3 Packet Directions

A simple flow property that can be observed in many protocols is the direction in which the first few packets in a session are sent. As previously mentioned in this report the obfuscated eDonkey protocols (both the UDP and TCP versions) do for example exhibit such a property, where packet 1 and 3 are sent by the client and packet 2 and 4 come from the server. These initial packets are in most protocols used to establish some common communication parameters, such as an encryption key or protocol versions, between the server and client.

We recognize the need to exchange this type of parameters at the beginning of a session, but this behavior need to be modified in order to achieve better obfuscation. In TCP based protocols we suggest that both the server and the client wait for a random duration after the completion of the TCP three-way handshake. Both server and client will then (for both TCP and UDP based sessions) start by sending padding, i.e. a blob of random size containing random data, before they initialize the intended communication. The purpose of waiting for a random duration before sending any data is in order to randomize whether the client or server sends the first packet in the session.

[1] There are some exceptions to this property though; applications that do not use TCP streams properly by presuming message boundaries to be aligned with TCP packet boundaries will not be able to handle randomized flushing [Davids, 2002]

7.3 Hiding Inside Well Known Protocols

Another possible technique for avoiding detection is to rely on already existing and widely used encryption techniques, such as SSL or SSH. This approach has for example been implemented for The Onion Router (TOR) network, where the traffic between each onion router pair is encapsulated with TLS [Dingledine et al., 2004]. TOR was, however, previously easily identifiable through its characteristic implementation of the TLS handshake. This flaw has been corrected in later versions of TOR, where the TLS handshake is designed to mimic that of Firefox+Apache [Dingledine, 2009].

Protocols that use UDP for transport are, however, not very suitable to tunnel inside SSL or SSH. The reason for this is that both these protocols utilize TCP for transport, i.e. they have properties such as guaranteed delivery and congestion control, which counteracts the purpose of using UDP in the first place.

An alternative to using SSL or SSH is to utilize the IPSec protocol for all communication between applications that need to communicate covertly. With IPSec we here refer to transport mode Encapsulating Security Payload (ESP), which provides an encrypted session between the two hosts. A desirable property of using IPSec, as opposed to SSL and SSH, is that it supports effective encapsulation of both UDP and TCP traffic. We believe the easiest way of controlling the IPSec session directly from the application is to encapsulate the IPSec packets using IPsec NAT-T over UDP 4500 as described in IETF RFC 3948 [Huttunen et al., 2005], doing so also solves the issue of having the IPSec packets traverse NAT firewalls.

As shown in [Dusi et al., 2009] simply just tunneling the protocol inside an encrypted tunnel might not be enough. The reason for this is that flow properties, such as packet sizes and inter-arrival times, can be identified even though the data has been encrypted. We therefore advocate that flow-level obfuscation techniques (such as those described in section 7.2) should be applied before tunneling the protocol inside some encrypted protocol, which only help to obfuscate payload properties.

Chapter 8

Conclusions

In this report, we have shown that obfuscated protocols, for which static pattern based fingerprints cannot provide accurate identification, can successfully be identified with a statistical approach. We present SPID, a statistical protocol identification framework that utilizes AttributeMeters to exploit payload and flow-based features of traffic sessions in order to identify the underlying application protocols. We first use a simple, greedy algorithm to produce sets of 8 and 15 AttributeMeters respectively, which are effective in differentiating between seven obfuscated protocols considered very hard to classify. We then look at each obfuscated protocol individually and highlight the most successful features for each protocol in order to provide insights in why protocol identification is possible despite obfuscation – information which will be valuable for designers of future obfuscated protocols.

One conclusion of our analysis approach is that the set of successful features (or AttributeMeters) for protocol identification depends very much on the mix of target protocols. Simply combining the AttributeMeters that work well for individual protocols gives worse performance than an optimization approach that considers the complete mix of target protocols to start with. While we believe that we can come up with a general multi-purpose set of AttributeMeters that can perform decently in most situations, we suggest to optimize the set on per-site and per-application basis. For network administrators this means that it is of importance to exactly specify the protocols which need to be identified and optimize the utilized set of AttributeMeters accordingly.

Another noteworthy general observation is that the AttributeMeters, which provide good classification results for the obfuscated protocols by encryption, are not only measuring flow properties. Some of the AttributeMeters that successfully identify encrypted protocols are in fact exclusively analyzing data in the TCP and UDP packets' payloads. The main reason for why statistical methods at all can be used to identify these obfuscated protocols is due to deficiencies in the obfuscation implemented by these protocols. Overcoming these deficiencies might not be easy though, at least not while maintaining requirements for high throughput without making the protocols overly complex.

In this report we provide two possible choices for enforcing network neutrality by deluding traffic classification engines. The provided options are forked into protocol obfuscation and tunneling inside well known encryption protocols. The protocol obfuscation fork seems to be the most popular path in today's P2P protocols, which seems to be working fairly well albeit the obfuscation flaws revealed in this report. The second fork, i.e. hiding inside a well known encryption protocol, is an alternative path we believe to be even more effective than protocol obfuscation. The reason for this is that the "hidden" traffic will look just like a normal SSH, SSL or IPSec session, which most network operators will be very reluctant to block or de-prioritize. It is, however, important to stress that even though encrypted tunnels provide for payload obfuscation, in order to avoid classification techniques such as [Dusi et al., 2009], one still need to apply flow obfuscation techniques also when tunneling traffic inside well known protocols.

8.1 Future Improvements of SPID

An important factor for the success of the SPID classification is the quality, but also the quantity and diversity of the available training data. It is therefore planned to get in contact with as many interested parties as possible in order to accumulate a database with Protocol Models with enough natural variation, creating new Protocol Models but also extending existing models with diverse training data from different network locations. Potential data providers can include academic institutions, network developers, private individuals or any other interested parties. Please feel free to contact the authors if you would like to contribute.

While we believe that we so far found a good set of AttributeMeters, we are certain that there is still room for improvement. This improvement can in the first place happen by adjusting existing AttributeMeters, but also by including newly created AttributeMeters exploiting additional flow or payload features of sessions. A complementary way of improving our set of AttributeMeters is to replace our simple, greedy AttributeMeter selection algorithm with more sophisticated approaches. Interesting candidates would be evolutionary algorithms, which could optimize SPID by randomly recombining and mutating sets of AttributeMeters, using classification performance (e.g. precision or F-measure) as "fitness function" between each consecutive evolutionary step.

In terms of improvements on the SPID framework, a first obvious improvement will be to consider the transport layer protocol (TCP or UDP) of sessions as a pre-filter, before actually applying the an adjusted set of AttributeMeters for each respective transport protocol. Furthermore, we believe that we could improve the results even more by implementing a service protocol cache, such as a Least Recently Used (LRU) cache that records the previously identified protocol for the most recent IP-and-port number combination. These improvements would not only increase the classification performance [Bar-Yanai et al., 2010], but also the performance in terms of execution speed. For usage on large-scale link, we are furthermore considering to remove short session from both training and validation data, as suggested in [Bar-Yanai et al., 2010]. In this way, we would sacrifice the completeness of the identification process for performance, which might be required to keep up with large traffic volumes in real-time.

A long-term goal is to be able to classify even traffic on high-speed backbone links [John et al., 2010b] in an online fashion. Protocol identification on backbone links will, however, require some further adjustments to the current SPID application besides speed improvements. As shown in [John et al., 2010a], routing symmetry on highly aggregated links is rare, which means that bi-directional session information can no longer be assumed and some AttributeMeters would need to be adjusted accordingly. We also believe that AttributeMeters disregarding packet payload would be even more important on backbone links, since payload inspection on large networks is often prohibited due to privacy concerns and legal implications [Claffy and Kenneally, 2010].

Acknowledgements

We would like to thank Sidney Amani and Cyril Jacob for the work they have put into this project. We especially applaud Sidney's efforts to collect training and validation data as well as having helped us with the automation of trace handling.

We would also like to thank Paul Gardner (of the Vuze development team), Eric Kollmann and Gunnar Kreitz for providing feedback on the contents of this technical report.

We are grateful to our funders, .SE (The Internet Infrastructure Foundation) and Chalmers University of Technology, for providing us the opportunity to conduct research in this field.

Finally, Erik would like to warmly thank his wife Sara for encouragement, advice and moral support throughout this project.

Bibliography

[Bar-Yanai et al., 2010] Bar-Yanai, R., Langberg, M., Peleg, D., and Roditty, L. (2010). Realtime classification for encrypted traffic. In *SEA*, pages 373–385.

[Bernaille et al., 2006] Bernaille, L., Teixeira, R., and Salamatian, K. (2006). Early Application Identification. In *ADETTI/ISCTE CoNEXT Conference*, Lisboa, Portugal.

[Biondi and Desclaux, 2006] Biondi, P. and Desclaux, F. (2006). Silver needle in the skype. In *Black Hat Europe'06, Amsterdam, the Netherlands, March 2006.*

[Bonfiglio et al., 2008] Bonfiglio, D., Mellia, M., Meo, M., Ritacca, N., and Rossi, D. (2008). Tracking down skype traffic. In *in IEEE Infocom08.*

[Bonfiglio et al., 2007] Bonfiglio, D., Mellia, M., Meo, M., Rossi, D., and Tofanelli, P. (2007). Revealing skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review*, 37(4):48.

[Callado et al., 2009] Callado, A., Szabó, C. K. G., Gero, B. P., Kelner, J., Fernandes, S., and Sadok, D. (2009). A Survey on Internet Traffic Identification. *IEEE Communications Surveys & Tutorials*, (3).

[Choi et al., 2004] Choi, B.-Y., Park, J., and Zhang, Z.-L. (29 Nov.-3 Dec. 2004). Adaptive packet sampling for accurate and scalable flow measurement. *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, 3:1448–1452 Vol.3.

[Claffy and Kenneally, 2010] Claffy, K. and Kenneally, E. (2010). Dialing privacy and utility: a proposed data-sharing framework to advance Internet research. *IEEE Security and Privacy.*

[Crotti et al., 2007] Crotti, M., Dusi, M., Gringoli, F., and Salgarelli, L. (2007). Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.*, 37(1):5–16.

[Davids, 2002] Davids, N. (2002). Tcp programming gotchas. `http://www.drdobbs.com/184416578`.

[despotify, 2010] despotify (2010). despotify — the open source spotify client and library. `http://despotify.se/`.

[Dewaele et al., 2010] Dewaele, G., Himura, Y., Borgnat, P., Fukuda, K., Abry, P., Michel, O., Fontugne, R., Cho, K., and Esaki, H. (2010). Unsupervised Host Behavior Classification from Connection Patterns. *Wiley IJNM: International Journal on Network Management.*

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[Dingledine, 2009] Dingledine, R. (2009). Tor and circumvention: Lessons learned.

[Dingledine et al., 2004] Dingledine, R., Mathewson, N., and Syverson, P. F. (2004). Usenix security symposium 2004. In *Tor: The Second-Generation Onion Router.*

[Do and Branch, 2009] Do, L. H. and Branch, P. (2009). Real Time VoIP Traffic Classification. Technical Report 090914A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia.

[Dreger et al., 2006] Dreger, H., Feldmann, A., Mai, M., Paxson, V., and Sommer, R. (2006). Dynamic application-layer protocol analysis for network intrusion detection. In *USENIX Security Symposium.*

[Dusi et al., 2009] Dusi, M., Crotti, M., Gringoli, F., and Salgarelli, L. (2009). Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 53(1):81–97.

[Erman et al., 2006] Erman, J., Arlitt, M., and Mahanti, A. (2006). Traffic classification using clustering algorithms. *SIGCOMM.*

[Gringoli et al., 2009] Gringoli, F., Salgarelli, L., Dusi, M., Cascarano, N., Risso, F., and claffy, k. c. (2009). GT: picking up the truth from the ground for internet traffic. *SIGCOMM Comput. Commun. Rev.*, 39(5).

[Haffner et al., 2005] Haffner, P., Sen, S., Spatscheck, O., and Wang, D. (2005). ACAS: Automated Construction of Application Signatures. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining network data*, page 202.

[Hjelmvik, 2009] Hjelmvik, E. (2009). Documentation of spid attribute-meters. *http://sourceforge.net/apps/mediawiki/spid/index.php?title=AttributeMeters*.

[Hjelmvik and John, 2009] Hjelmvik, E. and John, W. (2009). Statistical protocol identification with spid: Preliminary results. In *6th Swedish National Computer Networking Workshop (SNCNW).*

[Huttunen et al., 2005] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and Stenberg, M. (2005). Rfc 3948 - UDP Encapsulation of IPsec ESP Packets. RFC 3948 (Proposed Standard).

[Iliofotou et al., 2009] Iliofotou, M., Kim, H., Faloutsos, M., Mitzenmacher, M., Pappu, P., and Varghese, G. (2009). Graph-based p2p traffic classification at the internet backbone. In *IEEE Global Internet Symposium.* Citeseer.

[Internet Assigned Numbers Authority, 2010] Internet Assigned Numbers Authority (2010). Port numbers. http://www.iana.org/assignments/port-numbers.

[John et al., 2010a] John, W., Dusi, M., and Claffy, K. C. (2010a). Estimating Routing Symmetry on Single Links by Passive Flow Measurements. In *IWCMC'10: 6th International Wireless Communications & Mobile Computing Conference,.*

[John and Tafvelin, 2008] John, W. and Tafvelin, S. (2008). Heuristics to classify internet backbone traffic based on connection patterns. In *ICOIN.*

[John et al., 2010b] John, W., Tafvelin, S., and Olovsson, T. (2010b). Passive Internet Measurement: Overview and Guidelines based on Experiences. *Computer Communications*, 33(5):533–550.

[Karagiannis et al., 2005] Karagiannis, T., Papagiannaki, K., and Faloutsos, M. (2005). Blinc multilevel traffic classification in the dark. *SIGCOMM.*

[Kim et al., 2008] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. (2008). Internet traffic classification demystified: Myths, caveats, and the best practices. *ACM CoNEXT.*

[Kreitz, 2009] Kreitz, G. (2009). A glance at spotify's peer-to-peer streaming. *IETF 75, Jul 30 2009.*

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86.

[L7-filter, 2010] L7-filter (2010). Application layer packet classifier for linux. http://l7-filter.sourceforge.net/.

[Ma et al., 2006] Ma, J., Levchenko, K., Kreibich, C., Savage, S., and Voelker, G. (2006). Unexpected Means of Protocol Inference. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet measurement*, pages 313–326.

[Madhukar and Williamson, 2006] Madhukar, A. and Williamson, C. (2006). A longitudinal study of p2p traffic classification. *MASCOTS.*

[Mantia et al., 2010] Mantia, G. L., Rossi, D., Finamore, A., Mellia, M., and Meo, M. (2010). Stochastic packet inspection for tcp traffic. In *IEEE International Conference on Communications (ICC'10), Cape Town, South Africa, May 2010*.

[McGregor et al., 2004] McGregor, A., Hall, M., Lorier, P., and Brunskill, J. (2004). Flow Clustering Using Machine Learning Techniques. In *Passive and Active Measurement Conference (PAM)*, Antibes Juan-les-Pins, France.

[Moore and Zuev, 2005] Moore, A. and Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, page 60. ACM.

[Moore and Papagiannaki, 2005] Moore, A. W. and Papagiannaki, K. (2005). Toward the accurate identification of network applications. *PAM*.

[Nguyen and Armitage, 2008] Nguyen, T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys*.

[O'Brien, 2010] O'Brien, K. J. (2010). Skype in a struggle to be heard on mobile phones. *The New York Times*.

[O'Neil, 2010] O'Neil, S. (2010). Skype's biggest secret revealed. `http://www.enrupt.com/index.php/2010/07/07/skype-biggest-secret-revealed`.

[OpenDPI, 2010] OpenDPI (2010). Opendpi – the open source deep packet inspection engine. `http://www.opendpi.org/`.

[Pietrzyk et al., 2009] Pietrzyk, M., Costeux, J.-L., Urvoy-Keller, G., and En-Najjary, T. (2009). Challenging statistical classification for operational usage : the adsl case. In *IMC 2009, 9th ACM SIGCOMM Internet Measurement Conference, November 4-6, 2009, Chicago, IL, USA*.

[RFC 793, 1981] RFC 793, I. S. I. (1981). Rfc 793 - Transmission Control Protocol. `http://www.faqs.org/rfcs/rfc793.html`. Edited by Jon Postel.

[Rivest, 1992] Rivest, R. (1992). The rc4 encryption algorithm.

[Roughan et al., 2004] Roughan, M., Sen, S., Spatscheck, O., and Duffield, N. (2004). 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*, pages 135–148. ACM.

[Sen et al., 2004] Sen, S., Spatscheck, O., and Wang, D. (2004). Accurate, scalable in-network identification of p2p traffic using application signatures. *WWW*.

[Sommer and Paxson, 2010] Sommer, R. and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *IEEE Symposium on Security and Privacy*.

[Szabo et al., 2008] Szabo, G., Orincsay, D., Malomsoky, S., and Szabo, I. (2008). On the validation of traffic classification algorithms. *PAM*.

[Trestian et al., 2010] Trestian, I., Ranjan, S., Kuzmanovic, A., and Nucci, A. (2010). Googling the internet: Profiling internet endpoints via the world wide web. *IEEE/ACM Trans. Netw.*, 18(2):666–679.

[Vuze Wiki, 2010] Vuze Wiki (2010). Message stream encryption. *http://wiki.vuze.com/w/Message_Stream_Encryption*.

[Zhang et al., 2009] Zhang, M., John, W., Claffy, K., and Brownlee, N. (2009). State of the art in traffic classification: A research review. *PAM Student Workshop*.

[Zuev and Moore, 2005] Zuev, D. and Moore, A. (2005). Traffic classification using a statistical approach. *Passive and Active Network Measurement*, pages 321–324.