UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Facoltà di Ingegneria

Corso di Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXI Ciclo
Dipartimento di Informatica e Sistemistica

Understanding Internet Traffic: Can We Classify the Unknown?

Alberto Dainotti

Ph.D. Thesis

Tutor
Prof. Giorgio Ventre

Cotutor
Dr. Antonio Pescapé
Dr. K. C. Claffy

Coordinator
Prof. Luigi Pietro Cordella

November 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Motivation

Today, the Internet is a critical infrastructure of planetary scale, that affects almost every activity in the developed world. This was not foreseen in its original design, and for this reason new and increasingly urgent requirements (e.g. of security, reliability, billing, privacy, Quality of Service, etc.) are coming out. The Internet scenario becomes everyday more complex and constantly evolves driven by multiple uncoordinated actors. For example, we continuously assist to the appearance of novel applications and protocols, new-generation network architectures and infrastructures, as well as new paradigms of data communication (e.g. Peer-to-Peer) and user interaction (e.g. social networks). Because of all this, our current knowledge of the Network is poor. It is difficult to understand key aspects of the Internet, like its growth rate, properties of network traffic, the distribution of ISP interconnectivity, etc. [2]. Continuous measurements and analyses are therefore fundamental to understand, and to keep track of, Internet characteristics and phenomena, and they represent a starting point to influence such processes and to design the future.

In particular, the research field of Internet traffic measurement and analysis, has dramatically grown in the past ten years, also bringing contributions to several other fields of research on computer networks. Network traffic, in-

deed, is affected by all the events and interactions happening in a computer network. This makes traffic analysis a convenient point of observation to study computer networks. However, without appropriate tools and knowledge, network traffic appears a chaotic mix of data affected by multiple factors that are difficult to distinguish from one another. Since the first days of the Internet until today, the scientific community has made giant steps, adjusting techniques for traffic measurements, analysis methodologies, and modeling approaches. However, as mentioned, the Internet is not something still, neither network traffic. New problems arise, requiring either the development of new measurement and analysis approaches, or the revision of the current ones by taking into account issues like social aspects (e.g. data privacy), new communication paradigms (e.g. Peer-to-Peer), and the global scale on which events occur (e.g. distributed monitoring). At the same time, as mentioned above, requirements of better (i) security and reliability, (ii) control capabilities, and (iii) support to services offered by the network, are becoming more urgent. Network traffic analysis can make significant contributions to addressing such requirements.

For example, networking research has recently started dealing with a new problem that was unforeseen in the original design of the Internet: traffic classification. That is, the association of traffic flows to the corresponding network applications that generated them. Up to few years ago almost any Internet application was using well-known protocol ports that easily allowed its identification. This is not true anymore. The number of applications (e.g. Skype) using random or non-standard ports has dramatically increased. Moreover, often network applications are configured to use well-known protocol ports assigned to other applications (e.g. port TCP 80) attempting to disguise their presence. The initial answer to this problem was to resort to automated inspection of packet content through pattern-matching, in order to identify the application-level protocols. However this approach has several limitations. Firstly, there are privacy concerns regarding the access by providers to data transmitted by users. Secondly, since packet content in-

spection is a computationally-intensive task, with the increasing packet rates supported by network links it is becoming more difficult to deploy. Finally, payload inspection is not always reliable, and the crescent use of techniques like protocol encapsulation, traffic encryption, and protocol obfuscation, is making current techniques inadequate. For these reasons, the research community started to investigate alternative techniques for traffic classification based on traffic properties that do not require access to payload content.

In the last four years, dozens of scientific papers have been published, proposing a variegate set of classification approaches alternative to payload inspection, typically based on machine-learning techniques applied to statistical features of traffic. However several problems are still unsolved and in this thesis we address some of them. First, in practice there are no real implementations of the techniques proposed in literature, whereas science needs to produce prototypes that can be tested and evaluated on real traffic in order to validate such techniques and to demonstrate that viable alternatives to payload inspection exist. Indeed, because of the complexity and heterogeneity of the Internet, and because of its continuous change, new-generation instruments working on traffic properties must be tested in multiple contexts and with up-to-date traffic data. To fill this gap, in this thesis we present TIE, a community-oriented software platform for the development and comparison of implementations of traffic classification techniques. TIE has been developed also taking into account modern challenges that research on traffic classification must face, such as the need of *online* classification approaches (i.e. able to operate in realtime with live traffic) and the development of *multi-classifier* systems (i.e. that combine results from different classifiers).

Moreover, even when considering only results from published studies, a detailed examination of literature shows that up to now there seems to exist no perfect solution to the problem. All of the studies analyzed present some drawbacks or lack of complete evaluation. This suggests on one side the study of the combination of multiple approaches, on the other side, it urges to look for new techniques. In this thesis we propose a classification approach

based on statistical properties of traffic when observed at packet-level (i.e. in terms of packet size and inter-packet time). The choice of these properties is justified by previous studies, expounded in this thesis, in which we analyzed traffic at packet-level from different network applications, and we showed that it presents distinctive characteristics for each application.

Going further in the analysis of current literature, we observe that a large problem consists in the difficulty to compare the techniques proposed. This is because there are approaches working on different data, with distinct definitions, using different typologies of traffic classes and, finally, adopting different evaluation metrics. In the classification approach presented in this thesis we adhere to metrics commonly accepted and we include important metrics that several works have neglected (e.g. byte-accuracy of classification). Moreover, we designed TIE with the specific purpose to allow a rigorous comparison of different classification techniques. In this thesis it is shown how several design decisions have been made to allow comparison also among classifiers adopting definitions of classes, and of objects to be classified, that differ.

Finally, also thanks to TIE, we study the subject of classification through payload inspection from two points of view. On one side we try to understand if it is possible to remove few limitations held by payload-based approaches (privacy invasiveness and heavy computational load) in order to possibly combine them with more recent techniques. We show that it is possible to considerably reduce both computational load and access to sensitive data while keeping useful classification capabilities. Moreover, we study two payload-based approaches that represent the state of art in *ground-truth* systems (that is, systems for assigning reference classes to objects in order to evaluate a new classification approach) with the purpose to verify their reliability. Results obtained from recent traces show that today payload inspection presents several problems of accuracy and it can probably not be considered anymore as a stand-alone classification approach. A general conclusion that can be drawn from this thesis is indeed that traffic classification

in the future will probably resort to multi-classifier systems. The availability of TIE and the experimental results here shown, related to different approaches, can certainly serve as a contribution in this direction.

The next section illustrates the organization of the thesis, with specific references to each chapter.

1.2 Thesis Organization

The thesis is organized as follows. In the next chapter, we introduce network traffic analysis at packet-level. We explain the approach we use when studying the traffic generated by single applications, which is based on the search of invariants. That is, properties of network traffic at packet-level that are associated to specific applications and persist when traffic from the same applications is observed on different links and at different times. We then show several results obtained for different categories of applications: from more traditional ones (e.g. HTTP and SMTP), to more recent (games, Peer-to-Peer TV) and to malware (computer worms). In all cases we show how analysis at packet level reveals invariant properties and suggests several applications, as for example traffic identification and classification.

We introduce the subject of traffic classification in Chapter 3, starting from motivations and basic definitions. We present a detailed analysis of literature, and we conclude the chapter by highlighting several relevant open problems in this field that motivate the work presented in the next two chapters.

In particular, in Chapter 4 we present TIE, a software platform designed to fill several gaps in the field of traffic classification, most importantly the lack of actual implementations of classification techniques. We illustrate the components of the architecture highlighting the design decisions that make TIE focused on addressing important issues: comparison of classification techniques, multi-classification, *online* classification. Moreover we illustrate two first classifiers (implemented as TIE plugins) distributed with TIE: a port-based classifier and a payload-based classifier. These are used

as reference when studying other approaches in the subsequent chapter. We conclude Chapter 4 reporting on the connections between TIE and the networking research community, showing active collaborations and participation to international projects.

In Chapter 5 we present experimental studies regarding both machine-learning (based on statistical features) and payload-based approaches. Specifically, we first present a classification technique relying on features extracted from packet-level statistics. The most interesting aspect of this study is the novelty of the features considered, which despite the considerable amount of works published in literature, are completely new and possess powerful discriminating power. We then analyze a deep payload inspection technique to understand how much information from packets is really used in successful classifications. We make use of results from such analysis to design and evaluate a *lightweight* payload inspection approach with significantly lower requirements in terms of access to packet content and memory and CPU usage. Finally, we examine two payload-based approaches largely used in literature to establish ground-truth for traffic classification. Our experimental results reveal that they fail in identifying the entire traffic from the considered traces and that they often contradict each other. We draw conclusions regarding both the reliability of payload-based approaches in general and the design of better ground-truth systems.

Finally, in Chapter 6 we summarize findings and conclude the thesis.

Chapter 2

On Packet-level Analysis of Internet Traffic

2.1 Introduction

Internet traffic is studied by multiple points of view, each of them closely linked in terms of observed phenomena and of methodologies adopted. In this chapter we discuss the study of network traffic focused on network applications. Our purpose is to present findings we obtained when studying network traffic of various applications that are closely related to the contributions illustrated in the next chapters and pertaining to the field of traffic classification. It is our opinion indeed, that effective contributions to the field of traffic classification cannot be made without understanding properties of network traffic. Two of the objectives that drive our research in the field of traffic analysis are: (i) the search for peculiar properties of network traffic that are invariant with respect to the considered context (e.g. network link, time, etc.); (ii) the analysis of traffic from novel categories of applications emerged during last years (e.g. games, peer-to-peer, malware). We performed several studies by decomposing and observing traffic at several levels (e.g. aggregate link traffic, host-level, flow-level, packet-level), however in this chapter we focus on results obtained when looking at application traffic at packet-level, that is, in terms of *Inter-Packet Times* (IPT) and *Packet Size* (PS). This is for two reasons: (i) some of our most original contribu-

tions in the field of traffic analysis are related to packet-level analysis; (ii) part of the contributions to traffic classification presented in this thesis are actually based on a packet-level view of network traffic.

This chapter is organized as follows. In the next section we first introduce our approach to packet-level analysis of network traffic and we then present several results obtained when applying it to the traffic of: (i) classical Internet applications as HTTP and SMTP; (ii) novel applications as games and Peer-to-Peer TV; (iii) distributed malware like computer worms. The studies reveal interesting insights into the properties of network traffic that can contribute to several research areas and in particular to traffic classification. We conclude the chapter by summarizing, in Section 2.3, how the findings presented in the previous sections are related to the field of traffic classification and in particular to the contributions presented in Chapters 4 and 5.

2.2 Packet-level Traffic Analysis

Network traffic can be decomposed ad observed at different abstraction levels, some examples are: (i) in terms of aggregate link-traffic; (ii) at host level (iii) by using on-purpose definitions of sessions; (iv) at connection or flow level; (v) at packet level. Here we focus our attention on the packet level. Packet-level traffic characterizations express traffic flows in terms of *inter-packet time* (IPT) and *packet size* (PS).

We focus on a packet-level view of traffic because, when compared to higher-level approaches, it provides the following benefits: **(i)** traffic is observed at the deepest level of detail but at the same time observations are based on only two variables; **(ii)** working at packet level makes our approach independent of protocols evolution and applicable to different applications/protocols; **(iii)** switching devices often operate on a packet-by-packet basis, therefore realistic packet-level models are useful to evaluate their performance; **(iv)** most network performance problems (e.g. Loss, Delay, Jitter) happen at packet level, thus packet-level models are easily applicable

both to traffic simulations and synthetic traffic generations in order to study network-related issues;

The general approach we adopted in our study consists into analyzing statistical properties of traffic looking at it from different points of views: (i) we analyze properties of aggregate traffic, (ii) we decompose traffic into *sessions* (e.g. by source hosts, by flows, etc.), (iii) we measure sessions-related variables (as arrival times, size, duration, etc.), (iv) and we finally consider packet-level variables (IPT and PS) inside each session. When analyzing data, we look for repeating behaviors (the "*search for invariants*" [3]) and, by applying the same analysis to different applications, we aim at sketching similarities and differences. The adopted approach is indeed based on the observation of traffic both traversing different links and captured at different times.

Unless when differently specified, we captured and analyzed traffic using Plab [4], a software platform written in C, partially based on the Libpcap library [5], and freely available at [6]. Plab was designed to analyze traffic traces by identifying and storing data related to millions of *sessions* and to calculate and dump data ready to be processed by statistical analysis software. Also, more intelligence was introduced into the software, such as the ability to decode optional TCP headers like the MSS or to filter packets or entire sessions with several criteria. Depending on user-specified parameters, a session is identified by: (i) all packets sent and received by a host (*host mode*); (ii) all packets identified by source and destination IP and ports with a default timeout of 60 seconds (*flow mode*); (iii) as previous mode but by considering traffic in both directions (*biflow mode*); (iv) all packets exchanged by two hosts related to a specific service (e.g. TCP port 80), with a user definable timeout (*conversation mode*). Given one of the above modes, sessions are assigned an ID, and for each session the IPT between packets flowing in the same direction are calculated, along with PS. We call such data *packet-level* data series. An important aspect of our methodology is that in the evaluation of such data we usually do not take into account pack-

ets with empty payload. We indeed want to characterize the traffic generated by the applications, as much as possible independently of the transport protocols. By discarding packets with empty payload we drop all TCP-specific traffic, such as connection establishment packets (SYN-ACK-SYNACK) and pure acknowledgment packets [7]. For the same reason, in the estimation of packet size we measured the byte length of the transport-level payload. These choices make our results reusable for simulation purposes as an input for TCP state machines and UDP/IP stacks, like in D-ITG [6] and TCPlib [8]. Moreover, the arrival time of each session, its duration, and bytes transmitted for each direction are calculated by Plab, allowing to perform an analysis at a higher level (e.g. host/flow/conversation level). IPT and PS looking at traffic as a whole (aggregate link-traffic) are also calculated.

Part of the process of traffic analysis can be synthetically sketched into a number of sequential steps (with possible feedback lines) depicted in Fig. 2.1. After the acquisition of a traffic trace (first block), human intervention

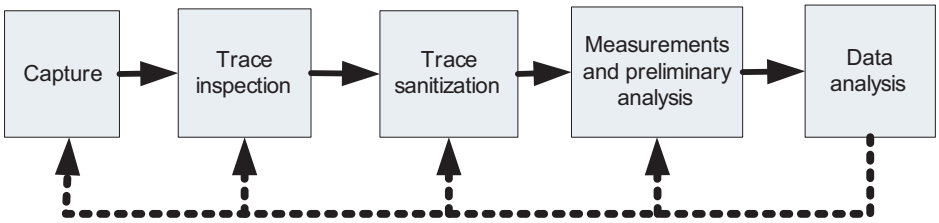


Figure 2.1: Life Cycle of Data Analysis.

is usually necessary to inspect it. Understanding which kind of traffic has been captured is a first fundamental step before performing a detailed statistical analysis. To do this, we need flexible and automated tools to rapidly investigate several properties of traffic, from looking into headers and payload (if present) to reporting concise information on hosts, flows, etc. From this analysis it is possible to choose the aspect on which we want to focus the characterization and to conceive strategies for automated trace sanitization to remove spurious data (step 3). In the next step, the software tool extracts measurements data from the traffic trace and it may also be able to perform a preliminary analysis. Finally, the data sets extracted by Plab

can be loaded into statistical analysis software (e.g. Matlab) and analyzed by looking at marginal distributions, time dependence, correlations etc. To this purpose we also developed a library of Matlab scripts, available at [6], which can be used for statistical analysis of traffic data, together with other tools made available by the research community.

2.2.1 An Overview of the Applications Considered

Here we introduce the applications analyzed in this section and we briefly discuss works in literature related to the study of their traffic.

HTTP. HTTP traffic has been obviously the subject of many research studies. The self-similar nature of HTTP traffic was stressed in [9]. It was proved that this property has a negative impact on network performance [10]. Self-similarity in HTTP traffic was explained by looking at the distribution of file sizes, transfer times, and inactivity times, which the authors studied by analyzing application-level data, as logs from clients and servers. The authors of [11] expanded the set of variables studied, partially based on heuristics to detect user clicks, by including: byte lengths of requests and replies, document size, user think time, etc. The authors of [12] applied a modeling methodology similar to that in [11] but tried to deal with the advent of the new HTTP/1.1 standard. They also introduced the concept of user *session*, to be used in the traffic generator they implemented. In 2004 the authors of [13] presented a new modeling approach based on TCP connections rather than on Web-page structure. They studied parameters as the inter-arrival time between TCP connections and, for each connection, the number of request/response exchanges, the delay between responses and subsequent requests, the number of bytes for each request and each response, etc. They also added a heuristic, similar to that of [11], to distinguish between requests of files belonging to the same document and requests of different documents. Not only most of the models proposed in literature are based on heuristics, but they are often relatively complex. The only study present at packet level

- at the time of our study - was [14], where a characterization related to the traffic of only a single PC in an Ethernet LAN was presented.

SMTP. As for SMTP, a message-level characterization was presented in [15]. They found that the trend of e-mail exchange follows user activity during the day and that the processes of arrivals and departures at servers are Poissonian in nature. In [16], in the context of the *ns* simulator, the authors proposed an email traffic characterization based on SMTP connection arrivals and bytes transferred per SMTP connection. Therefore, again, also for SMTP a packet-level characterization was not proposed up to the present work.

Multi-player Network games. Multi-player network games are rapidly becoming significant contributors to overall Internet Traffic and they are one of the most popular examples of real-time, interactive, and multimedia commercial Internet applications. The authors of [17] reported that about 4% of all packets in a backbone could be associated with only 6 popular games. In [18] it is also reported that the multi-player network computer games are predicted to make up over 25% of LAN traffic by the year 2010. Providing premium service to the increasing on-line gaming community may become a promising source of revenue for ISPs. Literature shows, indeed, that network games traffic presents different characteristics from other Internet traffic and therefore imposes different requirements on the underlying network. One of the first works related to traffic modeling of network games was presented by Borella [19]., providing an in-depth analysis of traffic traces from the popular multi-player first-person shooter game *Quake*. Empirical distributions of PS and IPT have been found and analytical distributions approximating them have been obtained through statistical fitting. In [20] Feng *et al.* describe results of the analysis of a 500 million packet trace of a popular on-line, multi-player, game server. They found that the behavior of the traffic generated by the server was highly predictable. They also found that the observed on-line games provide significant challenges to current network infrastructure

because of the presence of large, highly periodic bursts of small packets. The list of works examining network games traffic, analyzing them also at packet-level, incredibly grew in the recent years [21], [22], [23]. In the next section we show packet-level characteristics and invariant properties of *Counter-Strike*. As of May 2002, there were more than 20000 *Counter-Strike* active servers [24], and measurements from 2000 [17] indicated that the Half-Life/Counter-Strike application was generating a large percentage of all observed UDP traffic behind DNS and RealAudio.

P2P-TV. During last years, the research community has paid an increasing attention to the analysis of P2P IPTV scenarios, conducted with the aim to analyze the mechanisms of such systems, the traffic profiles, the perceived quality, and the behavior of the involved peers. This also entails new measurement approaches [25].
Sripanidkulchai et al. [26] showed that large-scale live streaming can be supported by P2P end-user applications despite the heterogeneous capacity of peers, paving the way to future studies in the field of P2P IPTV. Zhang et. al [27] presented the first measurement results about their protocol Donet [28], which was deployed on the Internet and called Coolstreaming. They provided network statistics, understanding of the user behavior in the whole system, and results related to the quality of video reception. In [29] [30] Hei et al. made a complete measurement of the popular PPLive application. They made active measurements by configuring their own crawler and providing many architecture and overlay details such as buffer size and number of peers in the networks. Based on their measurement studies, Hei et al. [31] developed a methodology to estimate the overall perceived video quality throughout the network. Vu et al. [32] made active measurements of the PPLive system and derived mathematical models for the distributions of channel population size and session length. Ali et al. [33] made passive measurements of PPLive and SOPCast applications and analyzed the performance and characteristics of such systems. Our work [34], from which we

present a packet-level analysis of P2P IPTV traffic in the next section, is different from those cited, because we focus on a set of four applications used worldwide and because of the live interest of the measured event.

Computer Worms. Computer worms have come to the public attention as small software, usually written by a single individual, capable to take down the Internet [35]. A worm is a small, self-replicating, portion of code which can rapidly spread over hundred thousands systems, generating an overwhelming amount of overall traffic and consuming huge computational power. In 2001 the Code Red I and II worms spread all over the world by exploiting a bug in Microsoft web servers, causing denial of services, systems and network compromise, and links overload, corresponding to several billions damage [36]. In 2003, the Slammer worm, the fastest worm ever, spread to 90% of all potential targets in less than 10 minutes [37]. The Witty worm, the first to carry disruptive payload, spread in March 2004. Ironically, it infected hosts proactive in securing their networks [38]. There is a rich literature of worm studies aiming at characterizing and modeling how the infections spread across networks, and of research works on worm detection and containment, based on many different approaches. On the other side, there is not much work related to a detailed analysis of the traffic generated by worms and comparing it to other traffic categories. To better understand related literature and how our contribution fits in, we can identify the following main areas: worm behavioral characterization, spread modeling, detection, traffic characterization. As for the first point, a comprehensive classification of computer worms is presented in [39]. Moreover, an analysis of specific worms (Witty [38], Slammer [40] [37], Code-Red [36]) is presented in several works. However, the results related to their traffic are basic: how and at which speed the worm spread, the scanning strategy and rate achieved, the distribution of IPs contacted, and how the packets are built. Sometimes the aggregate packet rate of worm traffic on a link is shown. Such information constitutes a valuable analysis of the worm characteristics,

which is a fundamental first step to understand worms, how they work and their impact, and to build deeper works on top of that. Another research field involves modeling the spread of worms using analytical and simulative approaches, also taking into account the effects of patching, human countermeasures, and congestion caused by the worms themselves (e.g. [41] [42]). Such studies can be used to design worm containment strategies. As regards detection techniques, they can be differentiated mainly in two approaches: content based and traffic based. Content inspection approaches can be based on signatures of known worms or on correlation of common patterns found in packets [43]. Content analysis, however, requires heavy resource consumptions and can be made uneffective by mutant worms. In contrast, detection methodologies based on traffic observations are related to the probing behavior of scanning worms. A common approach is to identify illegitimate scans and the increase of activity due to worm propagation [44] [45]. Relying on failed connections is obviously not viable for worms based on single UDP packets (e.g. Witty and Slammer). The works [46] [47] can be partially ascribed both to the research areas of detection and of worm traffic characterization. Because, starting from the observation of statistical properties of worm traffic - the exponential growth trend of infections at the early propagation stage - they propose two detection techniques. In the next section we present part of our work [48] on the analysis of Witty and Slammer, showing how from the traffic analysis of computer worms it is possible to derive important insights and elements for building traffic fingerprints.

2.2.2 Traffic Analysis of HTTP and SMTP

In this section we perform a packet-level study of HTTP and SMTP traffic in search of invariant properties by analyzing very large traffic traces captured from two WAN access links of a University and a research center, different in load, user population, as well as user practices. We captured traffic from two stub networks of academic and research Italian institutions during the period *Jan-Dec 2004*: (i) *Area della Ricerca di Genova of the Italian National*

Table 2.1: Details of observed sites and traffic traces

Site	Max Bandwidth	IPs	Date	Size	Pkts	Client-Server pairs	Conversations
UNINA	200Mbps	65000	14-19 Jul 2004	60 GB	830M	1M	2.3M
ARIGE	16Mbps	8000	4-10 Oct 2004	3 GB	43M	56000	125000

Research Council and (ii) *University of Napoli "Federico II"*. For brevity, we will refer to such sites respectively as ARIGE and UNINA. The observed links represent the only connection of the networks to the Internet and have a maximum throughput of $200Mbps$ and $16Mbps$ respectively. The observation of the links resulted in two traffic traces for each node: a trace related to the traffic generated by clients inside the observed networks reaching HTTP and SMTP servers on the rest of the Internet - we will call such traffic *Egress* - and the other one related to external clients visiting HTTP and SMTP servers hosted at the observed networks - we will call it *Ingress* traffic. Here we report results related only to Egress traffic, which represents the vast majority of the observed traffic and is related to a large population of users reaching all kinds of services. The analysis was performed on TCP packets with source or destination port 80 (HTTP) and 25 (SMTP).In order to preserve privacy, for each packet we kept only the IP and TCP headers and we scrambled IP addresses using the wide-tcpdpriv tool from the MAWI-WIDE project [49]. As regards HTTP traffic we also captured the first 3 bytes of payload data exchanged between each host pair (which under normal conditions correspond to the *method* invoked by the client in a HTTP request, see next section for more details).

Experimental results

We first decomposed network traffic into *conversations*, a concept previously introduced in literature [7], defined as the time interval during which two different hosts exchange packets belonging to an association of the kind {*source address, destination address, application-level protocol*}, and separated by a

fixed amount of time of silence. We defined as belonging to the same conversation, all packets to and from port TCP 80 (for HTTP) and port TCP 25 (for SMTP), traveling between two hosts, with an inactivity timeout of 15 minutes. As regards HTTP, it is worth to note that such a value is compatible with the HTTP session timeout introduced in past papers related to Web traffic [12] and the distribution of human thinking time modeled by Mah in [11]. They indeed represent an upper bound, being referred to the traffic exchanged between a single client and multiple servers instead of only a single server. Our approach does not take into account single TCP connections but considers all traffic happening during the conversation as a unique bi-directional flow of data, which we divided into *upstream*, which is traffic from the client to the server (packets with destination port 80 and 25 for HTTP and SMTP respectively) and *downstream*, traffic from the server to the client (the same condition as above but applied to the source port). Both for HTTP and SMTP we separately studied upstream and downstream traffic, building, for each of them, estimates of packet size and inter-packet time distributions. We want to stress that we calculate IPTs by subtracting the timestamps of two consecutive packets belonging to the same conversation and flowing in the same direction (upstream or downstream).

HTTP Traffic Characterization Details on HTTP traffic traces are reported in Table 2.1. We started studying observed links with a quantitative analysis of traffic. The three diagrams in Fig. 2.2 show rates for packets, bytes and opening of new conversations during all the week (from Monday to Saturday) at the UNINA site. Packet and byte rate are separated into upstream and downstream. The data has been sampled with a step of 15 minutes. We note diurnal patterns of activity, which is a known phenomenon in network traffic. Indeed traffic is mainly limited to working hours (9:00-17:00). Back to Fig. 2.2, it is interesting also to observe that bytes, packets and conversations tend to follow the same behavior keeping a proportion among them. Downstream traffic is several times higher than upstream traf-

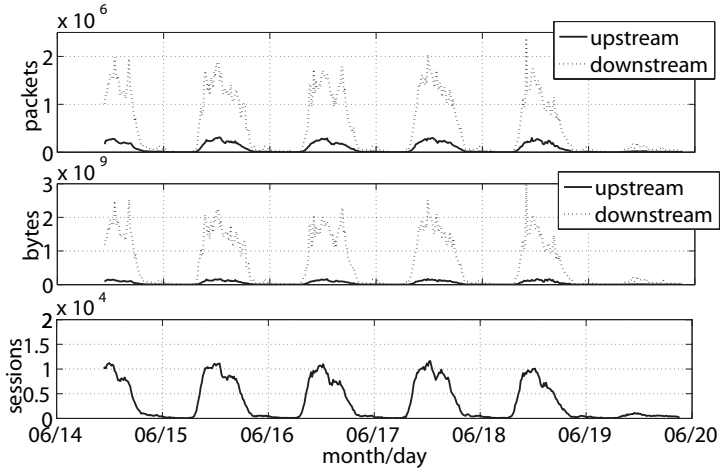


Figure 2.2: UNINA packet/byte/conversation rate

fic, both for packets (5:1 ratio) and bytes (15:1 ratio). Different ratios for packets and bytes anticipate what we will see later studying packet size distributions: upstream packets are usually smaller than downstream packets. All quantitative values observed at the ARIGE site are in accordance with those from UNINA keeping the same proportions among them but scaled by a factor of 20. The periodical patterns in traffic induced us to study payload size and IPT distributions separately for each day, from Monday to Friday. For each of the modeled variables (upstream/downstream – payload/inter-packet time), we found almost identical empirical distributions when comparing results for all the days of the week (an example for upstream inter-packet times is shown in Fig. 2.3). This is an important result of this study, because it proves time-invariant properties of the characterizations found. For each site we then built average distributions that were representative of the entire considered week. We also applied a statistical fitting methodology (not shown here but detailed in [50]) to such empirical distributions to come up with analytical representations of the sample sets. The weekly average empirical distributions were also used to compare PDF and CDF estimates of corresponding variables at each site. In the following, estimates of the distributions of upstream and downstream inter-packet time and payload size, found at both sites, are compared. It can be seen that all the modeled vari-

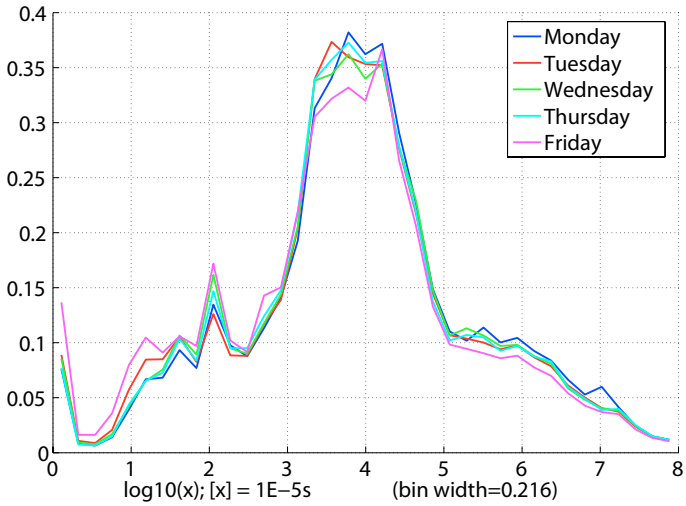


Figure 2.3: UNINA upstream inter-packet times PDFs

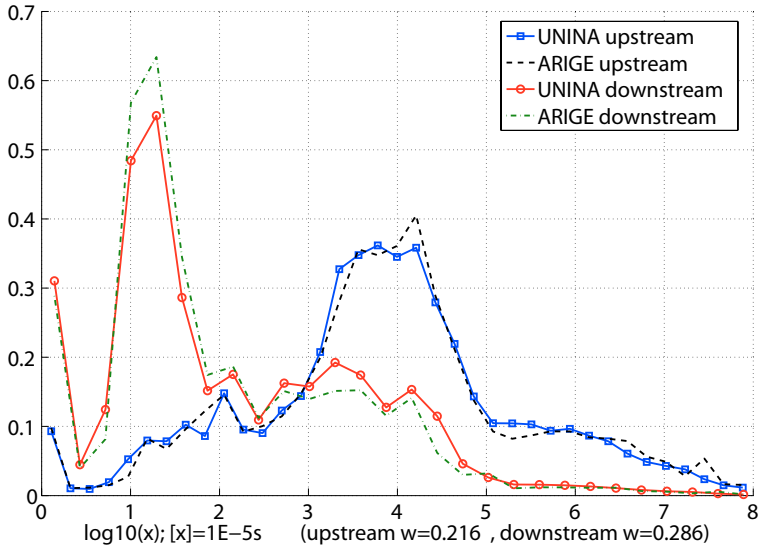


Figure 2.4: PDF of UNINA and ARIGE inter-packet times

ables show strong properties of invariance when the observed link changes. This is another fundamental finding of our work, which, together with the time-invariant characteristics cited above, makes the obtained results promising in terms of generalization capability and therefore applicable in realistic traffic generation and simulation. However, there are also some differences in the obtained models. Both analogies and dissimilarities, along with an analysis of the empirical and analytical models we obtained, are briefly exposed in the following.

As regards Inter-packet times, to properly read the plots in Fig. 2.4 we remind that packet timestamps have been collected with a precision of $10\mu s$ and that we applied a log_{10} transformation to the samples. We can ideally distinguish among three main overlapping regions in the inter-packet time distribution graphs. The lowest region, which can be roughly considered as starting from the first decade up to half of the third decade, is dominated by back-to-back packets probably due to file-transfers. The next region, which extends until about $1s$, contains samples which are compatible with RTTs found in Wide Area Networks. Finally, consistently with past works [9] [11] [12], we can assume that inter-arrivals directly generated by human behavior (e.g. user clicks) should be located in the last region, that is, beyond $1s$. Such scheme to read inter-packet time distributions has been cross-validated by several considerations and it can be observed that distributions have sudden changes in the transitions between one region and another. As we will also see for payload-size distributions, upstream and downstream traffic behave quite differently. Much of the samples in the upstream traffic is concentrated in the central region, while the majority of downstream inter-packet times are located in the first one. Indeed, while file transfers are the common task of HTTP servers, clients more often issue lot of requests. In the case of Web, for example, the first request of an HTML document is typically followed by more requests for the embedded objects. If such objects are small enough to be sent within one or few packets (as often is the case [51]), requests are sent with intervals close to the RTT from the client to the server. This, not only

justifies the predominance of the central region in the upstream distributions, but also explains its presence in the downstream distribution along with a correlation between upstream and downstream inter-packet times.

As regards PS, looking at Fig. 2.5 (in which CDFs are shown instead of PDFs, being that the diagrams are easier to be compared in this case), we can see that corresponding distributions from ARIGE and UNINA networks look very similar. Even though, we noticed that the Maximum Transmission Unit (MTU) of the network interfaces on the hosts, which limits the size of packets that can be sent and received, has a partial effect on them. Indeed, we found several spikes in the distributions. To confirm the hypothesis that they were not related to specific application or user behaviors, but were caused by MTUs set on the interfaces, we instructed Plab to decode, when present, the Maximum Segment Size (MSS) TCP option in SYN segments. We were then able to build separate payload distributions for conversations in which peers negotiated a different MSS (based on the smallest MTU on their interfaces). An analysis of such distributions, and a comparison against the original global distribution revealed that the peaks in the latter were generated by packets with maximum payload from conversations in which the MSS of one of the peers limited the size of the payload. Even if the four most frequent negotiated MSS (1460, 1380, 512, 536 bytes) were found to be associated to more than 95% of the total packets, with the first two covering more than 90%, their distribution was not totally invariant from site to site. Indeed, a larger percentage of conversations with MSS=1380 bytes is the reason of the slight difference between UNINA and ARIGE downstream CDFs.

As we anticipated, upstream and downstream packets have different payload-size distributions. In general, upstream payloads tend to be smaller, with a mean value of 500 bytes; average downstream payload size is 1240 bytes. The downstream distribution is basically dominated by full-length packets. We identify several of the above-mentioned peaks, which together sum up to the 80% of the distribution. Most of them are concentrated near 500 bytes and

near 1400 bytes. The remaining 20% of the packets is uniformly distributed over the total range. If we remind that we discarded packets without payload, we see that such distribution is consistent with the trimodal distribution of generic IP packet size found in wide-area traffic studies [52]. For upstream traffic we found totally different results. 85% of the samples reside in a set starting from 120 bytes to 1280 bytes.

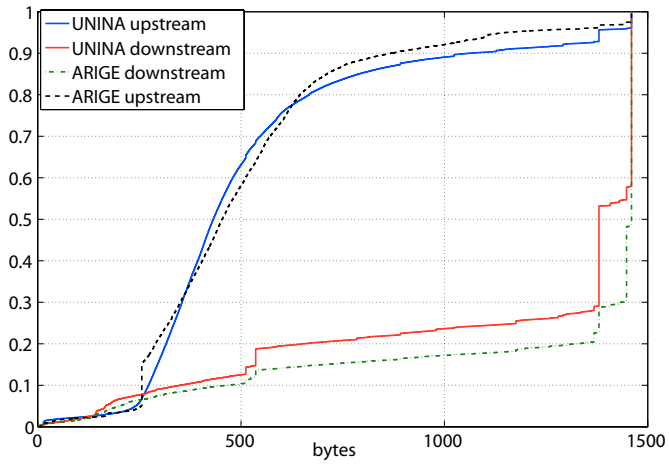


Figure 2.5: Upstream and downstream payload size CDFs

Impact of P2P traffic on port TCP:80 on traffic profiles. To stress the concept of spatial invariance at packet level, in this Section we show how a process of trace sanitization applied to the UNINA traffic makes the traffic profile more regular and more similar to the profile of the ARIGE network traffic. As a side effect, we give some insights about non-HTTP traffic using port TCP 80. Indeed, we know that few applications are sometimes configured to use port TCP 80 to bypass filters on firewalls. Especially P2P applications [53]. For this reason, to enforce network usage policies, network administrators sometimes must resort to application-level firewalls or to more advanced filtering architectures. Latest firewalls and routers, indeed, are moving toward the use of packet inspection, often done in hardware, to enforce traffic filtering by application classification; e.g. the NBAR feature in Cisco routers [54]. An NBAR filter was indeed in action on the UNINA

site, which blocked all communications with hosts outside the network using FastTrack and Gnutella protocols. This prevents some popular P2P applications (Kazaa, Grokster, Limewire, Morpheus etc.) to exchange traffic. At the ARIGE site there were no such kind of filters instead. This network, though, is used by a more homogeneous and restricted group of users, mainly researchers, than the large UNINA network. In the latter, different categories of people (e.g., students) have access to the network. Before starting our analysis, we were still concerned with the presence of non-HTTP traffic in our traces, so we investigated it further. We added a feature in Plab to examine the first 3 bytes of the first packet carrying TCP payload exchanged in each conversation. As reported in Table 2.2, we observed that almost 94% of the conversations started with a GET request, 4% with a POST request etc. Only a small fraction of the sessions presented packets starting with a byte not corresponding to an alphabetic character. Inside this category, 99% of the conversations started with the byte 0xe3. As reported in [55], this is the first byte exchanged by peers opening a communication session based on the eDonkey2000 protocol (used by eDonkey and eMule file-sharing applications). Because our interest was in characterizing traffic generated only

Table 2.2: Payload inspection on first packet opening a conversation

Conversation Start	GET	POS	HEA	Downstream	0xe3	PRO
Percentage	93.94	4.23	0.7	0.44	0.27	0.2

by applications running over HTTP, we instructed Plab to recognize such conversations and to filter them out. Also, 0.44% of the conversations were initiated by the host communicating from port TCP 80 (labeled as "downstream" in Table 2.2). By filtering our traces, we observed that 5.12% of the processed packets were discarded. Therefore, this non-HTTP traffic represents a not negligible portion of the captured traffic. As regards the number of filtered conversations, they account for about 0.7% of the total. This suggests that filtered conversations tend to generate more packets than HTTP

conversations. By comparing the results obtained with and without filtering such conversations, we observed that discarded traffic had a consistent impact in terms of payload size and inter-packet time. Comparisons of the obtained distributions for upstream traffic at the UNINA site are shown in Fig. 2.6. Observing the properties of such distributions it is clear that fil-

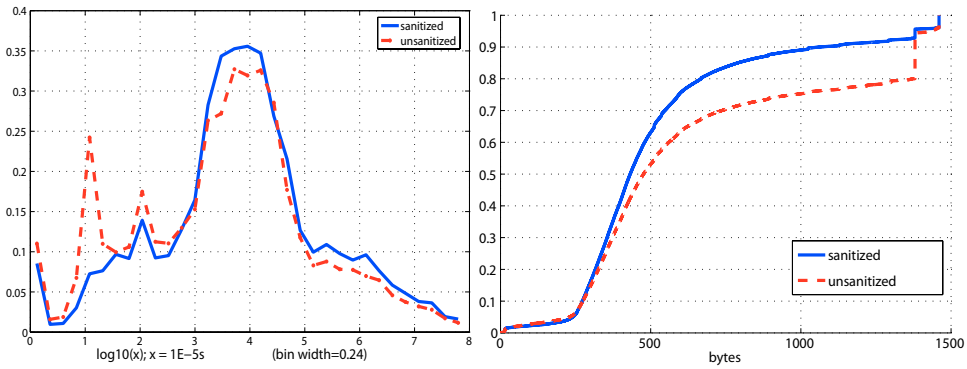


Figure 2.6: Filtered UNINA upstream: IPT PDF (left), PS CDF (right)

tered conversations increase the portion of back-to-back packets with full payload, probably due to the presence of file-transfers.

SMTP Traffic Characterization. In this section we present results on SMTP showing that looking at another type of traffic using the same packet-level methodology highlights how a different type of traffic exhibits different properties. Indeed, by comparing traffic from ARIGE and UNINA networks, also in this case we found a spatial and temporal independence. Therefore, as for the SMTP traffic we briefly show results of the same approach applied to traffic related to port TCP 25, captured at UNINA in the same days as the HTTP traffic trace. The diagrams in Fig. 2.7 show indeed a totally different behavior from HTTP traffic, and instead reflect the mechanisms of the SMTP application-level protocol. Servers (mail receivers) tend to send small packets with inter-packet times mostly concentrated around $300ms$. The CDF diagram shows that about 90% of the payloads are smaller than 100 bytes and almost all payloads do not go beyond 300 bytes.

On the contrary, looking at upstream traffic we find that clients (mail

senders) behavior is dominated by large back-to-back packets. Also, in the payload-size distribution we found spikes corresponding to negotiated MSS, as explained above. The jump at 1380 bytes is easily recognizable in Fig. 2.7.

The observed distribution characteristics are strongly related to the application. Indeed in the SMTP protocol the mail receiver answers to commands and data submissions with a rigid syntax in which replies have a numeric code, therefore they are usually very short. Mail senders, after issuing commands, must instead transfer all the mail content, prepended by headers, and often with binary objects attached. On the other side, knowing the application characteristics, it is evident that, unlike for HTTP, there is not much user interaction involved. However, user influence can be seen at least with the presence of long emails and file attachments. The latter, for example, lead to a higher percentage of maximum-size packets (data transfer) compared to small-size ones (possibly SMTP commands). This aspect can also be observed in conversations data, such as conversation duration and transferred bytes. Once again, this reveals a correlation between these two observation levels.

Discussion

In this work we applied a packet-level analysis both to HTTP and SMTP traffic. We selected HTTP traffic because it is a traditional Internet application-

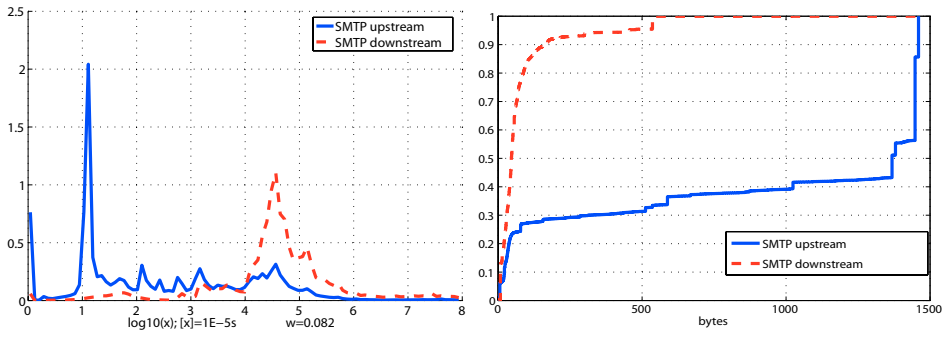


Figure 2.7: UNINA port TCP 25: IPT PDFs (left), payload CDFs (right)

protocol and despite a lot of works about HTTP traffic modeling and characterization are present in literature, few papers are devoted to traffic characterization at packet level. Therefore, the study of HTTP gave us a twofold opportunity. Firstly, we provided a characterization of a well established and largely diffused protocol. Secondly, thanks to the large number of papers studying HTTP (not at packet level) we compared, contrasted, and understood the results we obtained with a packet-level characterization. Furthermore, in order to show how our approach is simply usable in the case of other protocols and how a packet-level approach can be useful to understand traffic peculiarities we analyzed also SMTP traffic. This experimental analysis shows that packet-level characterizations exhibit invariant properties, in terms of time (*temporal invariance*) and observed link (*spatial invariance*). We can summarize what we obtained from our analysis in the following points: (I) packet-level models show invariance with respect to space and time; (II) payload size and inter-packet time distributions of upstream traffic are very different from those related to downstream; (III) MSS can influence the payload-size distribution; (IV) thanks to the sanitization of traffic traces we found confirmation that P2P traffic using port TCP 80 to bypass firewalls significantly changes the traffic profile on this port;

2.2.3 Traffic Analysis of a Network Game

In this section we investigate the statistical properties of IPT and PS of a well known client/server network game, *Counter-Strike* (CS). Specifically, by analyzing its traffic under very different network scenarios we look for invariant properties specific to this traffic. In the first subsection we show results obtained through a comparative analysis between results we obtained from traffic traces made available by the scientific community, and a study present in literature. Whereas in the following subsection we show results obtained by repeating several experiments with a Counter-Strike server and a variable number of clients under a heterogeneous set of network scenarios. Both studies show how the traffic profiles of this game exhibit strongly invariant

properties.

Searching invariants in available data

In this subsection we show a brief comparative analysis of the same game in two different contexts. More precisely, first we perform a characterization from a WAN traffic trace collected at server-side and made available by the authors of [20] (see Table 2.3). Then, to study the invariant properties of CS traffic, we compare our results against those from the analysis performed in [1], which is related to LAN traffic collected at client-side. We show how it is possible to see packets Inter-Arrival Times (IAT) at the server, as the aggregation of Inter-Departure Times (IDT) from a number of clients, even if the traffic has been captured in two contexts that are totally different.

In the case of the WAN scenario, traffic was captured at a Counter-Strike server of one of the most popular on-line gaming communities in the Northwest region of USA, mshmro.com [56]. The server is configured with a maximum capacity of 22 players (which is the average number of players in fact). While in this trace the point of observation is from the server, the other considered scenario is related to traffic captured at client side in a LAN environment. The results from the second scenario, which we used for the comparison, have been presented in [1]. In that work Faerber, by analyzing empirical distributions of IPT and PS of packets flowing in and out clients, proposed a simple model of Counter-Strike traffic.

Table 2.3: LAN and WAN Traffic Trace Details

Scenario	Observation point	Log Time	Packets
WAN	Server	7h:50m	20000000
LAN	Client	5h:3m	284519

After the characterization we performed for the traffic captured in the WAN scenario, we focused on the search for invariants between the two considered traffic traces by comparing the incoming/outgoing traffic of the server in one trace against that of the clients in the other trace. In Fig.2.8 we show the PDFs of both the IPT observed at client-side (IDT) in the LAN scenario and the IPT of aggregate client traffic captured at server side (IAT)

in the WAN scenario. We found that the average value of the IDT samples reported in [1] - 41.7 ms which correspond to ca. 23 $pkts/s$ - is consistent with the mean of the server IAT (from all the 22 clients) that we measured: 0.002 $s \longrightarrow$ 506 $pkts/s = 22\ players \times 23\ pkts/s$.

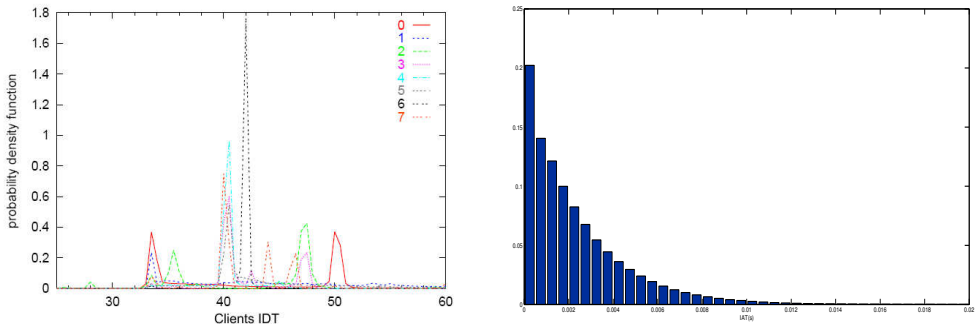


Figure 2.8: IPT PDF: client-side IDT (left) [1], server-side IAT (right)

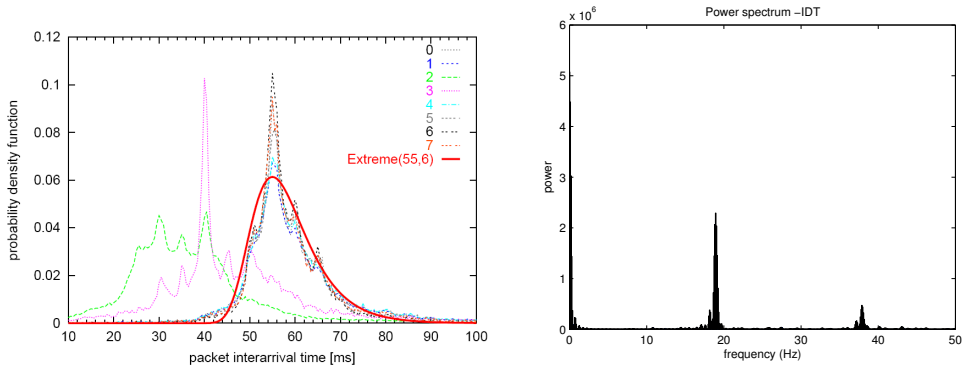


Figure 2.9: Client-side IAT PDF (left) [1], Frequency spectrum of Server-side packet rate (right)

In Fig.2.9 the PDFs of the IAT at clients (in the LAN scenario), and the frequency spectrum of the packet rate of server-to-clients packets (measured in the WAN scenario) are depicted. From the frequency analysis we found a notable peak around 19 Hz, probably connected to regular game updates (ca. each 53 ms) sent from the server to the clients. Indeed, looking at the PDFs in the left diagram we see that most distributions are centered around 55 ms.

Experiments with a heterogeneous testbed

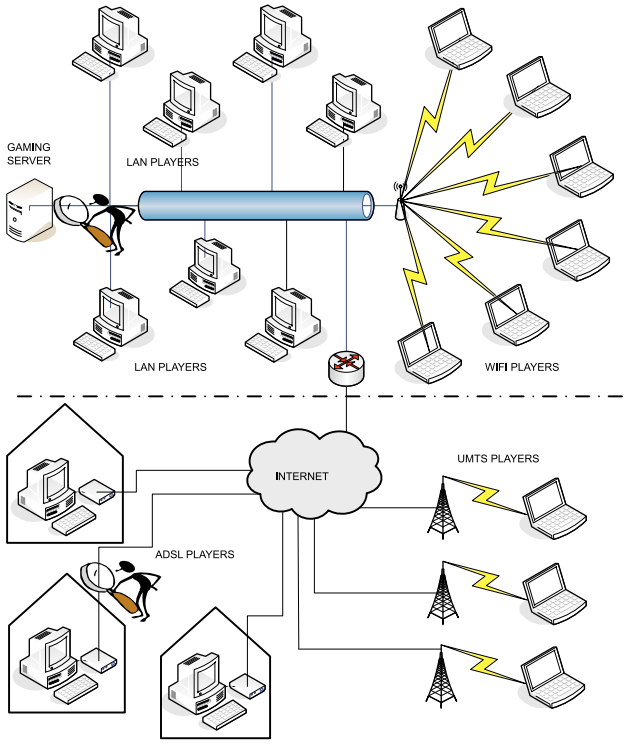


Figure 2.10: Experimental testbed

To further investigate the foreseen invariants examined in the previous subsection we collected more data. To this aim a heterogeneous testbed, schematically depicted in Figure 2.10, has been set up. We placed the game server into a LAN. Then, we made tests with clients on LAN, Wi-Fi, Home ADSL, with different configurations and number of users. We performed a characterization of the traffic captured in all the experiments, and we found strong invariants reported in the following. We report here some of the results of the characterization for the simple LAN scenario, in which four players (Luca, Alessandro, Biagio, Roberto) played for about 20 minutes being directly connected to the same LAN where the server was placed. In tables 2.4 and 2.5 we show some summary statistics for this scenario. Figures 2.11 and 2.12 show respectively the four CDFs of the PS generated by the four clients in upstream traffic, and the four PDFs of IPT upstream packets.

Table 2.4: UPSTREAM PS statistics

player	mean [bytes]	std [bytes]	iqr [bytes]	max [bytes]	min [bytes]	median [bytes]	entropy [bits]
luca	42.49	7.99	14.00	61.00	25.00	44.00	4.62
alessandro	43.81	8.41	15.00	61.00	25.00	46.00	4.73
biagio	43.69	8.11	14.00	61.00	25.00	46.00	4.66
roberto	45.66	7.58	12.00	60.00	25.00	49.00	4.49
server	77.48	32.79	32.00	461.00	27.00	72.00	6.67

Table 2.5: UPSTREAM IPT statistics

player	mean [μs]	std [μs]	iqr [μs]	max [μs]	min [μs]	median [μs]	entropy [bits]
luca	41910	7134	171	465432	16730	41687	2.39
alessandro	42876	16783	443	1522475	3668	41716	4.04
biagio	42175	12245	363	1528904	31225	41724	3.65
roberto	41861	1195	264	63509	32183	41728	2.78
server	13912	9975	26720	42177	3	13894	3.22

We can observe how the distributions are similar for all the players. The distribution of PS from clients is made of very short packets, usally in the [25,61] bytes interval and mostly concentrated at 50. The packets sent by the server are usually larger (approximately in the [1,460] bytes interval), gamma-like distributed, and with higher variance. As for IPT of packets generated by the clients, they are gaussian-like and centered at $\simeq$ 42 ms. Figure 2.13 shows the autocorrelation of upstream PS for all the four players, revealing in all cases a high degree of correlation up to the 5th-6th packet which slowly decades.

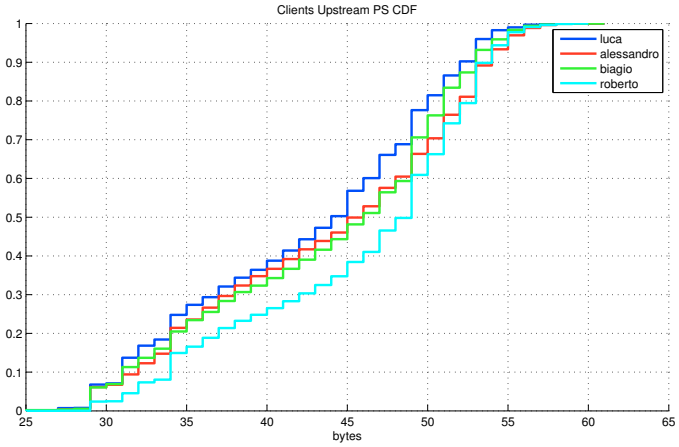


Figure 2.11: LAN scenario: CDF of clients upstream PS

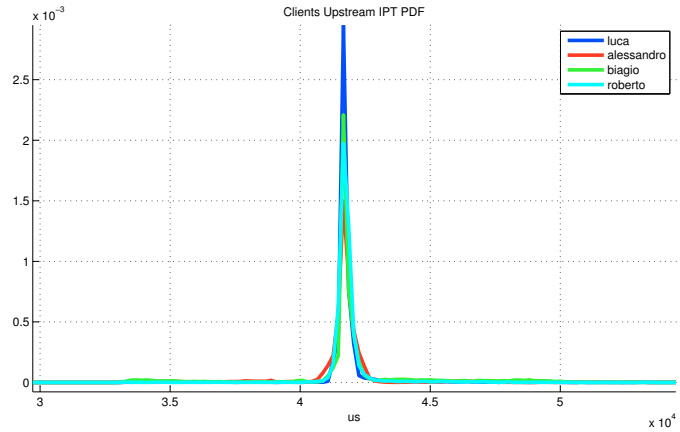


Figure 2.12: LAN scenario: PDF of clients upstream IPT

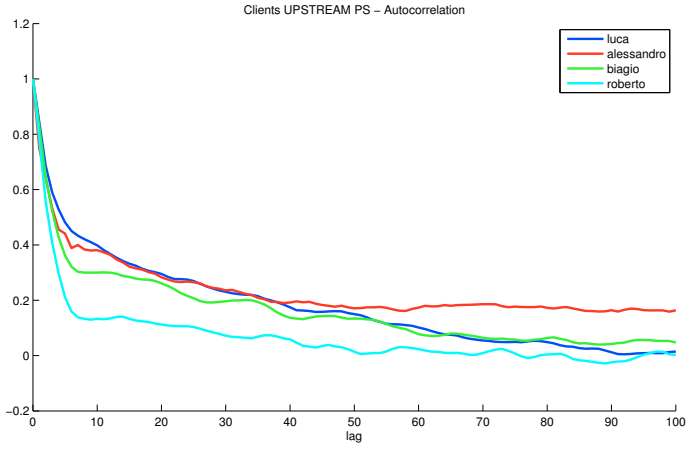


Figure 2.13: LAN scenario: Autocorrelation of clients upstream PS

To show an example of invariant properties with respect to the access network, in Figures 2.14 and 2.15 the four PDFs of IPT upstream packets are shown for network scenarios respectively with users connected through a wireless LAN and users at home connected through an ADSL connection. The IPT PDF is therefore very characteristic of the game, independently of the access network to which the clients are connected.

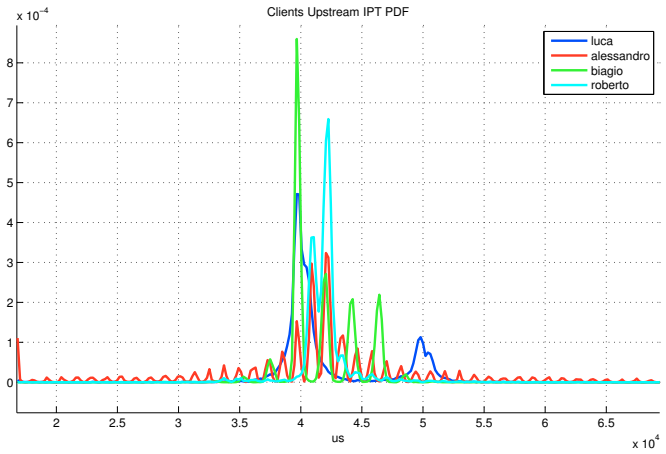


Figure 2.14: WLAN Scenario: PDF of clients upstream IPT

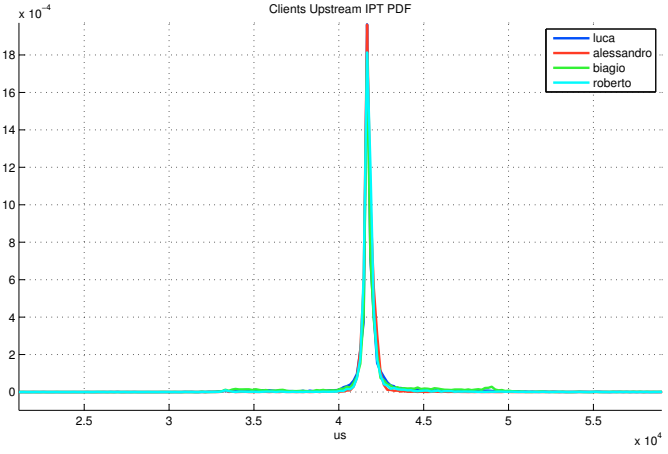


Figure 2.15: ADSL Scenario: PDF of clients upstream IPT

Discussion

The search for invariants allowed us to identify the behaviors related to the application by isolating them from the specific network environment. All clients have the same behaviors, both in terms of IPT and PS. PS have shown to be totally invariant with respect to the network, whereas IPT are strongly invariant but become more spread when the packets traverse a WAN. In conclusion, we can state that the traffic of Counter Strike is highly predictable. Allowing, e.g., ISPs to identify it, and to provide specific QoS and billing accordingly.

2.2.4 Traffic Analysis of Peer-to-Peer TV

In this section we point our attention on the study of P2P IPTV applications. More precisely, we study the traffic generated by the four most used P2P IPTV applications at the time of the experiment, and still considered today among the top ones: *PPLive*, *PPStream*, *Sopcast*, *TVants*. Analyzing four applications instead of a single one makes our analysis more complete and allows to investigate the generalizability of the observed results and to look for invariants of this traffic category.

The overall study that we performed on this traffic ([34][1]), of which here we present only excerpts related to packet-level analysis, aims at a better understanding of the mechanisms used by such applications and their impact on the network, despite their use of proprietary unpublished protocols, by directly looking at the traffic they generate. We aim at understanding: (i) which transport-level protocols are used and what are the consequences of different choices; (ii) how traffic is divided into signaling and data, and into upload and download directions, in order to study and characterize them separately; (iii) criteria useful to discriminate between signaling and data traffic and to identify P2P IPTV traffic; (iv) statistical properties of P2P IPTV useful to understand the impact on network nodes and links (e.g. long

[1]This work was developed jointly with Université Pierre et Marie Curie - Paris 6 in the context of the CONTENT EU NoE

range dependence); (v) how peers interact, how much they contribute to content distribution, and what is their typical lifetime; and (vi) what is the download policy of the different applications. The results presented here are particularly relevant to identify traffic generated by such applications, to understand their impact on network nodes and links, and to build realistic simulations and emulations.

Description of the experiments

With the aim to better understand both traffic properties and peer behavior of a P2P IPTV community during a worldwide event, we considered four applications. Analyzing different applications allows studying such communities without being too closely related to the design of the applications and thus making the results more general. We collected traffic traces during the 2006 FIFA World Cup from June 09 to July 09 because we believe that it can be representative of events of interest in P2P IPTV communities. The 2006 FIFA World Cup represents indeed one of the biggest worldwide sport events that attracted tens of millions of viewers from all over the world. For our experiments, we chose the applications PPLive, PPStream, SOPCast and TVAnts, because they are among the most popular. Actually their users, on the community website at [57], ranked these applications among the best and efficient applications to watch live television. Nowadays, these four applications are still very popular and, e.g. in the case of PPLive [58], estimates indicate millions of users. All the largely deployed P2P IPTV systems claim to use a mesh-based architecture as those investigated in this work. The mesh-based architecture used by P2P IPTV systems takes its inspiration from BitTorrent [59] and uses the same kind of swarming protocol, as in Donet [28]. Instead of building a strict topology (e.g. a broadcast tree), a mesh is built among peers whose links (peering relationship) depend on the data availability on each peer. The topology is dynamic and will continuously evolve according to the peering relationship established between peers. These P2P protocols generate two kinds of traffic: video traffic which

is used for exchanging data chunks, and signaling traffic used for exchanging the information needed to get the data. Thanks to the signaling, peers know how to download the video data chunks by exchanging randomly with other peers information about the data chunks they have (buffer map) and the neighboring peers they know. Therefore, with such signaling traffic, each peer discovers iteratively new peers and new available data chunks. However, even if these applications are freely available and developers state to use a mesh-based architecture, their source code is not open and their exact implementation details and protocols are still widely unknown. Therefore, we can only count on traffic analysis to understand their transmission mechanisms and peer behavior.

We collected a huge amount of data, measuring most of the World Cup soccer games with four different applications at the same time. Here we focus on four packet traces, one for each application, collected on June 30 in the campus network of the Université Pierre et Marie Curie - Paris 6. From our collection, we selected these traces because on that day two very important quarter-final matches were played, which attracted a lot of P2P IPTV users. The traces are publicly available at [60]. It is worth stating that we also analyzed the other collected traces and we obtained results similar to those presented in this work .

On the selected day, two quarter-final matches were scheduled: *Germany vs. Argentine* in the afternoon and *Italy vs. Ukraine* in the evening. The choice of this day was motivated by non-technical issues too: to have the highest number of users involved in the trace we collected, we considered matches with favorite teams, team of the hosting country, etc. During each match, we used two computers, each one running a distinct P2P IPTV application as well as `WinDump`[61] to collect the traffic. Therefore we collected two traffic traces for each match, one for each application. In particular, we respectively collected traffic from PPStream and SOPCast during the the first match and from PPLive and TVAnts during the second one. To collect packets from our measurement testbed we used two PCs equipped with

1.8GHz CPUs, common graphic card capabilities, and running `Windows XP`. The PCs were situated in the campus network and were directly connected to the Internet through a 100Mbps `Ethernet` link. For all the measurement experiments, the consumed bandwidth was always relatively low and did not exceed 10Mbps. The Ethernet cards did not suffer any packet loss and captured all the packets. For all the experiments, the nodes were watching *CCTV5*, a Chinese TV channel available for all the measured applications. It was important to watch the same TV channel with all the applications to assure that the behavior of peers was similar in each trace. For example, despite the different applications, during the advertisements a user may stop watching the channel switching the application off and then switching it on a few minutes later. After collection, the traces had to be cleaned by removing packets not related to the applications. This operation was necessary because we did not know the characteristics of the traffic of such applications. Therefore, we first captured all the traffic exchanged by the nodes under test. After that, we inspected the traces and filtered out traffic not related to the observed applications. This task was done both manually and using Plab [62].

Experimental Analysis

In this section we analyze traffic characteristics in detail. In particular, we first describe some general properties of this traffic, then we discuss issues related to the separation of video and signaling flows, and we show distinct results for them. The considered applications generate traffic using different ports and protocols. Table 2.6 contains the information regarding the used protocols and the sizes of the traces. The time duration of the collection ($\approx$ 225 minutes) is longer than that of a soccer match ($\approx$ 105 minutes). We chose to collect the traffic before and after the games to capture all the effects that the live interest on a soccer game could produce on the behavior of peers (e.g. flash crowds). We observe that there is much more traffic in the upload direction (i.e. from our controlled node to the other peers) than in the

Table 2.6: Summary of packet traces.

	PPLive	PPStream	SOPCast	TVAnts
Duration (s)	13,321	12,375	12,198	13,358
Size [MB]	6,339	4,121	5,475	3,992
Download[%]	14.11	20.50	16.13	24.76
TCP	14.09	20.50	0.23	14.71
UDP	0.02	≈ 0.00	15.90	10.05
Upload[%]	85.89	79.50	83.87	75.24
TCP	85.81	79.50	3.89	61.67
UDP	0.08	≈ 0.00	79.98	13.57

download one (i.e. from all the other peers to our node). This is due to the fact that our computers are connected to the Internet through a 100 Mbps Ethernet link. Therefore, in contrast with more common ADSL connections, we have equal upload and download capacity. This implies that we are able to provide video chunks to a large number of peers. Interestingly, we can notice that PPLive, TVAnts and PPStream make extensive use of TCP, whereas SOPCast runs mainly on UDP. Moreover we can observe that TVAnts also relies on UDP for a non negligible percentage of packets. Table 2.7 shows the ports used by the applications. PPLive and SOPCast present a similar behavior. Indeed, for these applications, the machine under test uses mostly the same ports for all the communications with the other peers which, in turn, use a wide range of different ports. PPStream behaves similarly, except that it uses a fixed remote port and three different local ports for the very few UDP packets. It is also interesting to note that both PPStream and PPLive

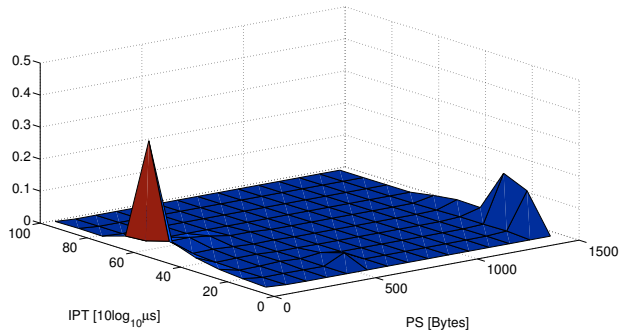
Table 2.7: Utilized Port number (percentage of packets).

		PPLive	PPStream	SOPCast	TVAnts
Remote Peers	TCP	Several	Several	Several	16800 (> 25%)
	UDP	Several	7201 (100%)	Several	16800 (> 60%)
Controlled Peer	TCP	10549 (> 99%)	11430 (> 99%)	8516 (> 99%)	16800 (> 71%)
	UDP	5747 (100%)	5747 (42%), 11430 (54%), 65535 (4%)	8516 (> 99%)	16800 (> 99%)

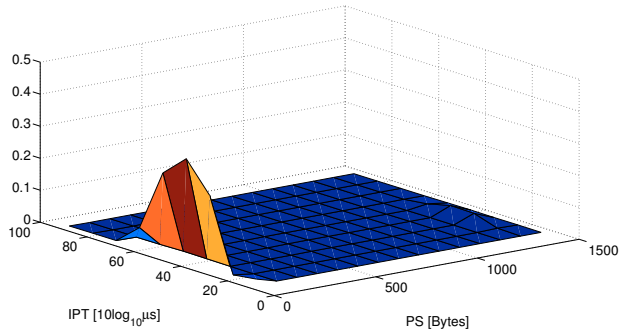
use the local UDP port 5747. Finally, a peculiar behavior is noticed for TVAnts, which uses port 16800, both local and remote, for most of the UDP and TCP packets. This is probably because TVAnts sets a default port on a new installation that can be changed thereafter by the user. Looking at Table 2.7, it is also evident how P2P IPTV traffic cannot be reliably identified by looking at transport protocol ports, motivating the need to find different ways to recognize their traffic.

As we explained in the previous section, the P2P applications we studied generate two kinds of traffic: video and signaling. The signaling traffic of P2P IPTV systems is not expected to be delay-sensitive, because it is used to exchange information about peers or data availability but is not used for interactive commands, as for Video on-Demand systems like Joost [63]. In video on-demand systems, the users may want to move the video playback instant forward or backward promptly. In the case of P2P IPTV, it is not possible to have this kind of interactive commands since the data flows are broadcast live. In general we can say that the signaling and video traffic have not the same characteristics such as packet size or delay constraints, and they would have a different impact on the network. Therefore we want to separate video and signaling traffic in order to analyze their peculiar properties.

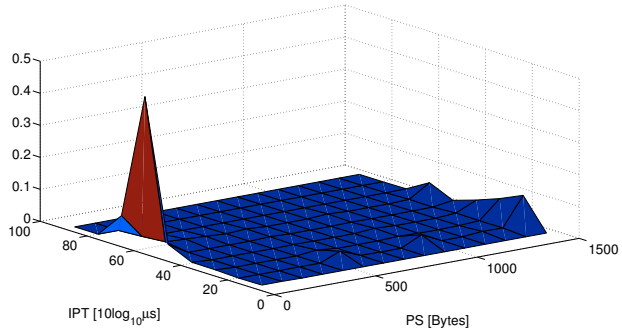
Because the protocols adopted by such applications are not open, we rely on a heuristic based on traffic properties. A simple heuristic to separate these two kinds of sessions in PPLive traffic was previously proposed by Hei [29]. Such heuristic works as follows: for each session (same IP addresses and ports), we count the number of packets larger than or equal to 1200 Bytes. If a session has at least 10 of such large packets, then it is labeled as a video session. All the non-video sessions are supposed to carry signaling information. To understand if it was reasonable to apply such heuristic to all of them, we investigated traffic properties for all of the four applications, driven by the following considerations. It is expected that video sessions are essentially composed of large-sized packets sent at small and regular time intervals, whereas signaling information should be carried by smaller packets



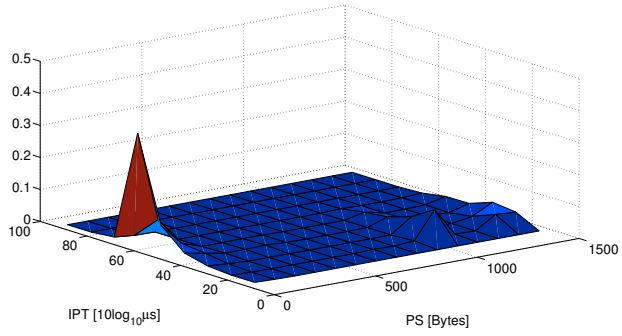
(a) PPLive



(b) SOPCast

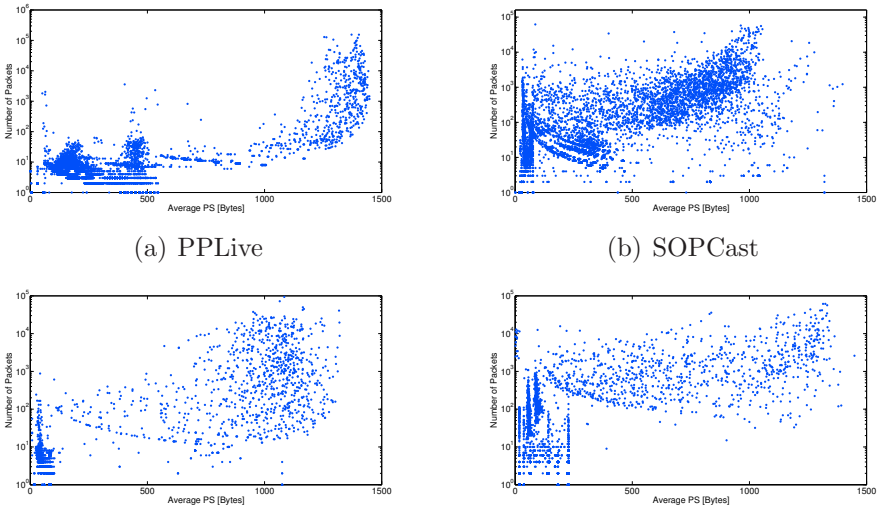


(c) PPStream



(d) TVAnts

Figure 2.16: Joint probability distribution of Inter Packet Time and Packet Size.



(a) PPLive (b) SOPCast

(c) PPStream (d) TVAnts

Figure 2.17: Upstream flows: Average Packet Size vs Number of Packets.

sent much less often compared to video chunks. For the same reasons we expect to find that signaling sessions exchange much less packets than video sessions in general.

Figures 2.16 and 2.17 reveal interesting properties of overall P2P IPTV traffic generated by the four considered applications. Moreover, they confirm the above intuitions by showing that there are packets and sessions with different properties and that the packet size property may be a good heuristic to discriminate between signaling and video sessions.

Figure 2.16 shows the joint Probability Density Function (PDF) of the Inter Packet Time (IPT) and Packet Size (PS) of the download traffic. The IPT of each packet is the time elapsed between that packet and the previous one of the same session, and as usual for the PS we considered the protocol-layer payload size, discarding all TCP packets without payload. For each application we only considered packets related to the prevalent transport protocol, e.g. TCP for PPLive and UDP for Sopcast. The distributions of these applications are different but, for all of them, we can distinguish two main clusters of packets: small-size packets (< 200 Bytes) with large IPT and

large-size packets (> 1000 Bytes) with small IPT. Most of the video packets should then belong to the large PS and small IPT cluster. The signaling packets, instead, should mostly belong to the other cluster with small PS and large IPT.

In Figure 2.17 instead, we show scatter plots in which, the coordinates of each point are given by the average PS and the number of transmitted packets of a session. The number of transmitted packet is plotted on a logarithmic scale axis. In these diagrams the sessions with the largest numbers of packets (supposedly video sessions) tend to have high average packet size. Both these results made us very confident that the cited heuristic could be used for all the P2P IPTV applications considered. Furthermore, to be sure that this heuristic does not introduce large errors in our analysis, we also manually inspected the traces. This verification allowed us to discover that there are different kinds of signaling packets, that such packets have fixed sizes, and that these sizes are always smaller than 1000 Bytes. Thus, considering also the findings about the PS distributions of the four applications, we modified the heuristic to use a limit of 1000 Bytes instead of 1200 Bytes. Finally, we can state that, with regard to the traces we consider, the heuristic proposed is effective in discriminating signaling and video traffic.

In Table 2.8 we report statistics on the ratio of signaling traffic with respect to overall traffic of all the applications, also separated in download and upload. We observe that Sopcast is by far the application producing more signaling traffic, whereas PPLive generates much less signaling than the others. In all the four cases the amount of signaling traffic we sent is much smaller than that we received. This can be explained by observing that we sent a large quantity of video chunks.

Discussion

In this section we analyzed the network traffic generated by four of the most popular P2P IPTV applications. Such applications use proprietary unpublished protocols, making their study challenging. However this work goes

Table 2.8: Signaling Traffic Ratio

	PPLive	PPStream	SOPCast	TVAnts
Total	4.1%	13.6%	19.3%	10.2%
Upload	2.2%	10.8%	13.6%	7.8%
Download	19.2%	25.8%	48.5%	18.0%

into the direction of improving knowledge of current P2P IPTV systems. We think the results here presented can be useful in several fields: (i) to identify traffic generated by such applications; (ii) to understand the impact of their traffic on the networks; (iii) to build realistic simulations and emulations.

We outlined similarities and differences among such applications in terms of the transport layer protocols and the related ports they use, deriving some interesting properties, e.g. which applications run only on TCP and which ones rely also on UDP, and showing that such traffic cannot be identified by using port numbers. The first step to understand and identify P2P IPTV traffic is to discriminate between signaling and data traffic. We discovered several properties of the traffic that strongly confirm, for all the applications considered, a heuristic (previously proposed in literature only for PPLive and with slightly different parameters) to separate signaling and data sessions. This step was fundamental to further analyze operation and exchange of traffic in P2P IPTV communities [34]. Moreover, we gained some knowledge regarding statistical properties of this traffic (e.g. PS-IPT distribution, recurring PS, etc.) that in the future we plan to further investigate as means for application identification through traffic analysis.

2.2.5 Traffic Analysis of Computer Worms

In this section we present the analysis and characterization of the Witty and Slammer worms. By studying and comparing traffic from three network links we show interesting properties of time and space invariance and some peculiarities of worm traffic which make it different from other categories of traffic commonly found on the Internet. Besides representing the first

step into understanding worm traffic more deeply, such results can be also considered for the design of new fingerprinting and detection techniques.

The Witty worm [38] exploited a bug in the ISS firewall software when it decodes ICQ servers packets [38]. Witty sends a single UDP packet with source port 4000 to each scanned host. The payload varies from 768 to 1279 bytes because of a random padding which is done to make worm identification (e.g. by firewalls) harder. After that 20000 packets have been sent to randomly chosen IP addresses, it overwrites a small portion of the hard disk, and then it starts to send packets again. The Slammer worm [40] [37] instead, exploits a bug in Microsoft SQL Server. It sends a single UDP packet of fixed size (404 bytes) with destination port 1413 to each target. The scanning strategy is random. However, a bug in its random number generator left a considerable portion of the Internet hosts not scanned. Differently from other worms (that are latency-limited because they issue a *connect()* call for each host to be scanned), as for example Code Red, both Witty and Slammer are bandwidth-limited worms. This is because they send UDP packets and do not need to wait for any response from the potential victim. So they are only limited by the bandwidth of the infected machines.

In Tab. 2.9 the traffic traces that have been used in this work are summarized. As for the Witty worm, we analyzed several tens of gigabytes of data collected and made available by CAIDA [64]. The traffic stored in such files has been collected by a network telescope, that is, all the traffic directed towards an unused address space has been captured. This way, unsolicited traffic (e.g. automated scans) can be detected and observed. The traces here used have been obtained by filtering the traffic captured by the network telescope, in the days of the spread of the Witty worm, considering only UDP packets with source port 4000. Moreover, to obtain more traces related to Witty and to overcome the poor availability of worm traces, we looked into traffic traces of a trans-oceanic link during the days of the worm spreading, verifying the presence of packets which can be associated to the Witty worm (second row of Tab. 2.9). Indeed, the MAWI-WIDE project [49] makes avail-

able 15 minutes traffic traces of this link for each day of the year since 2000. An important benefit of such approach is that we also have the availability of data related to legitimate traffic captured from the same link, and at the same time, of the worm related traffic. This is good to compare their properties. As explained in Section 2.2.5, many results from the analysis show that the Witty traffic selected from this trace has consistent properties with those from the trace made available by CAIDA (evidence of spatial invariance).

We also looked into the MAWI traces that were captured during the spread of Slammer. But we could not find packets associated to this worm. This is probably due to a filtering rule which was set on the routers. Traffic traces related to Slammer have been made available by MIT [40]. They were obtained by filtering all the traffic traversing two unidirectional links, considering only UDP packets with port 1413. These traces have been collected on March 25th, 2004, which was one of the days of highest Slammer activity. All the traces used contain only packet headers until layer 4, that is, no payload information is stored.

Table 2.9: Traces details.

Worm	Source	Observation point	Filter	Date	Duration	Size	Infected Hosts
Witty	CAIDA	Net Telescope	udp src port 4000	March 20-22, 2004	15m per day	1.3GB	10725
Witty	MAWI	BIDIR Link	ALL	March 20-22, 2004	15m per day	2.1 GB	4728
Slammer	MIT	UNIDIR Link 0	udp dst port 1434	March 25, 2003	8h 44m	842 MB	2523
Slammer	MIT	UNIDIR Link 1	udp dst port 1434	March 25, 2003	8h 44m	431 MB	5321

Analysis of Witty Traffic. We consider IPT inside host-based sessions (i.e. an IPT represents the time between two consecutive packets sent by the same host). We measure IPTs with a resolution of $1\mu s$ and apply a logarithmic transformation because they range over several orders of magnitude. For each trace the distribution of IPTs is built by putting together all the IPTs calculated for each host-based session. In the following, when necessary, we will refer to the corresponding PDF as the *average* PDF, to distinguish it from the PDF made by IPTs of a single session.

In Fig. 2.18 the diagrams of both the CDFs and PDFs of IPTs are

depicted. The distributions are quite regular, resembling a gaussian distribution. This behavior is invariant with respect to both the site observed and the time of observation. As for the first point, the mean of the distribution is shifted when the link changes. This can be probably connected to the number of IPs that can be observed: the lower the number of victims per host, the larger the average IPT. As for the *invariance* with respect to time, we observe that the IPT distribution derived from a specific observation point does not change in the different epidemical stages. Indeed, as can be seen from Tab. 2.10, while the first day represents the explosion of the epidemic, in the subsequent days the infection level decreases dramatically, probably because of patching (see infection models taking patching into account [42]). Also, Tab. 2.11 shows that, for each site, all the distribution statistics but the entropy[2] keep approximately the same values as the considered day changes. In contrast, the entropy follows a descending trend as the infection decreases, possibly because the number of infected hosts decreases thus reducing the *uncertainty* associated to the PDF. This finding suggests a possible application to identify the evolution status of a worm spreading.

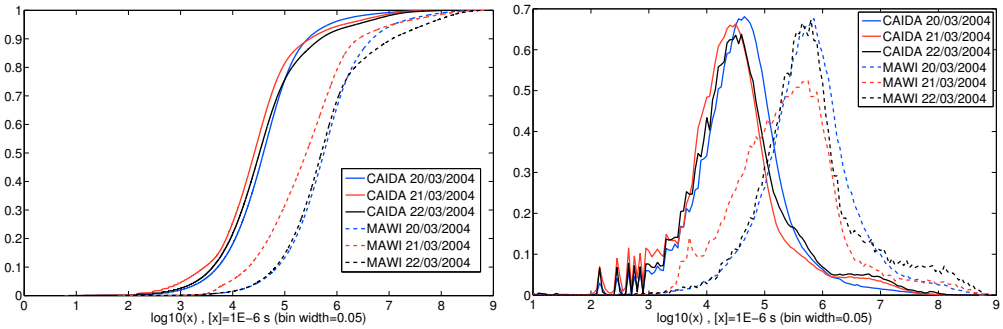


Figure 2.18: Witty Inter-packet times.

Another interesting aspect that came out from the study of Witty traffic is related to the payload size. As anticipated, this worm is designed to pad the packet payload with a random number of bytes. In Fig. 2.19 the CDF and PDF diagrams of the PS of packets sent by Witty hosts from both CAIDA

[2]The entropy is calculated as $-\sum_i P(x_i) \cdot \log_2 P(x_i)$ where $P(x_i)$ is the probability associated to each bin (of width 0.05) of the samples histogram.

Table 2.10: Witty Traffic statistics.

	CAIDA 20/3	CAIDA 21/3	CAIDA 22/3	MAWI 20/3	MAWI 21/3	MAWI 22/3
Pkts	9.261.414	2.986.325	701.314	226.034	102.727	33.941
Src Hosts	7.515	2.128	1.085	2.881	1.141	706
Dst Hosts	6.800.779	2.690.668	683.096	198.663	99.380	33.231

Table 2.11: Witty IPT $(log10(x), [x] = 1E-6s)$.

Trace	Mean	Median	Max	StdDev	Entropy (bit)
CAIDA 20/3	4,593	4,459	8,870	0,750	7,825
CAIDA 21/3	4,454	4,415	8,845	0,873	7,247
CAIDA 22/3	4,592	4,513	8,717	0,885	6,570
MAWI 20/3	5,762	5,750	8,898	0,720	6,112
MAWI 21/3	5,413	5,410	8,904	0,902	5,710
MAWI 22/3	5,802	5,670	8,921	0,902	5,118

and MAWI traces of three different days are depicted and compared. The figures show that in all cases the distributions can be well approximated by a uniform distribution from 768 to 1279 bytes, which is totally different from the typical payload size distributions commonly found on Internet links [52].

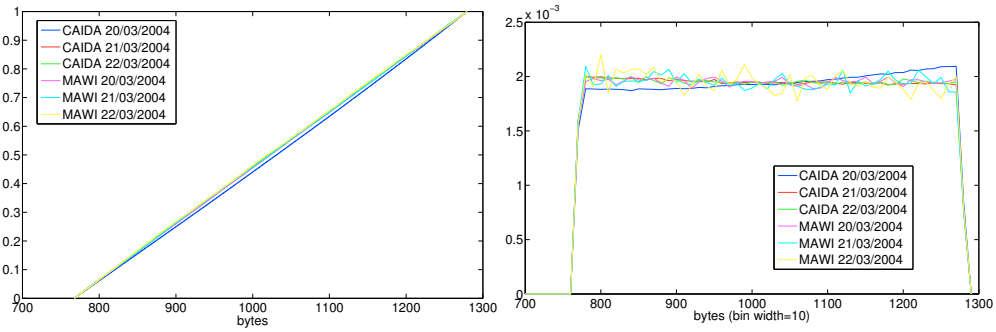


Figure 2.19: Witty Payload Size.

A joint PDF diagram displays information related to the PS and IPT marginal distributions, taking into account also mutual dependencies between the two variables. In Figures 2.20 and 2.21 the joint PDF diagrams of Witty traffic, related to MAWI and CAIDA are respectively shown. They are very similar, confirming a typical behavior, from a traffic characterization point of view, of the infected hosts. This is an interesting invariant, which makes such diagrams (or the information contained) to be considered for fingerprinting and detection techniques.

To understand how much the traffic properties we found are really pecu-

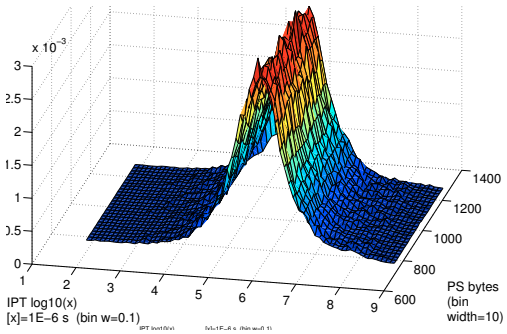


Figure 2.20: MAWI: Joint PDF (20/03/2004).

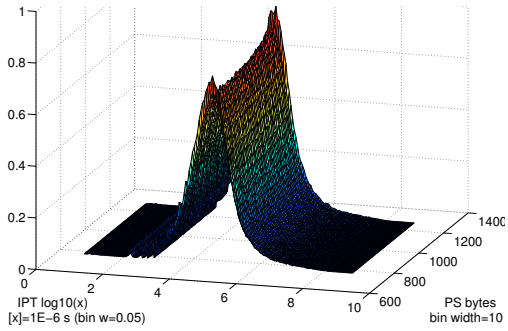


Figure 2.21: CAIDA: Joint PDF (20/03/2004).

liar to Witty, we made some comparisons against legitimate traffic. In this process, in order to make comparisons more meaningful, we tried to remove all possible differences due to side-effects. For this reason, we chose DNS traffic as an example, because it runs on the same transport protocol - UDP - of Witty (in contrast, TCP end-to-end flow control could somehow affect packet-level variables) and a DNS server, like a worm-infected host, talks to several different hosts in a short time. Moreover, the DNS traffic analyzed is from the same MAWI trace of Witty, therefore there are no link-dependent or time-dependent aspects which could be differently influenced. In Fig. 2.22, the joint PDF of PS and IPTs calculated for the 4593 DNS servers (hosts sending packets only from source port UDP 53) found in the MAWI trace of 20th of March shows a totally different profile. The DNS packet payloads are rarely larger than 250 bytes, with a stronger concentration around three byte-lengths, and IPTs are spread but with the main peaks in the first decade and in the region between the fourth and seventh decades. To stress

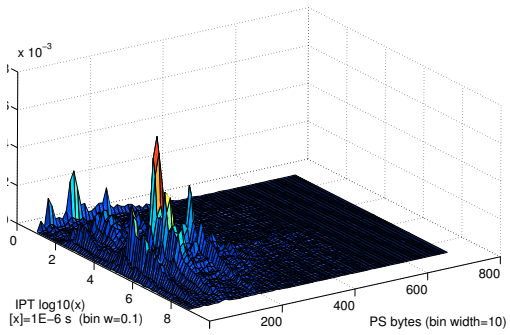


Figure 2.22: MAWI DNS joint PDF.

the concept of the possible application of such findings in the field of fingerprinting and classification, in Fig. 2.23 we show the joint PDFs obtained for two single hosts. On the left, one of the Witty-infected hosts, chosen among those generating most traffic, is shown. Whereas the diagram on the right is related to the most active DNS server. The choice of a larger binning and the less smoothness of surfaces are due to a reduced number of samples compared to the PDFs obtained by averaging data of all hosts. However we can

see that: (i) the average joint PDFs reflect the properties of the single hosts, and (ii) the joint PDFs of the two considered hosts are totally different.

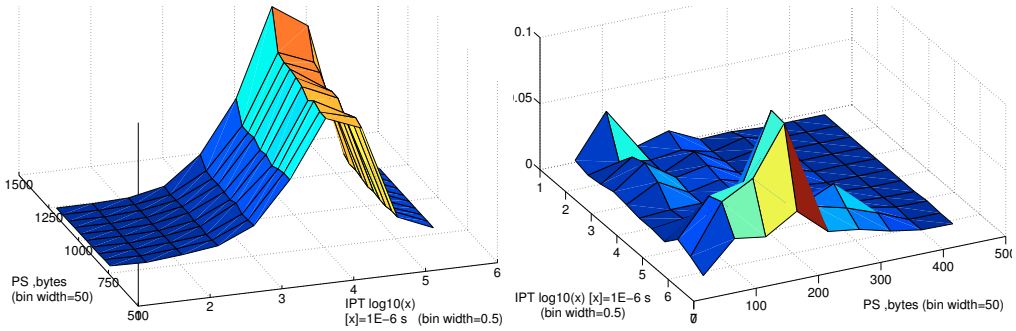


Figure 2.23: Joint PDFs of single hosts: Witty (left) and DNS (right).

Besides looking at marginal distributions, we also studied PS time dependence. For several hosts infected by Witty, we plotted the PS autocorrelation function from lag 0 to 100 (Fig. 2.24), and compared them to the corresponding ones generated by DNS servers found in the MAWI trace. Such graphs clearly highlight the different behavior of a *Witty-infected* host from a legitimate host (a DNS server). The sequence of Witty PS is totally uncorrelated, whereas there are strong indications of correlation in DNS traffic. The uncor-

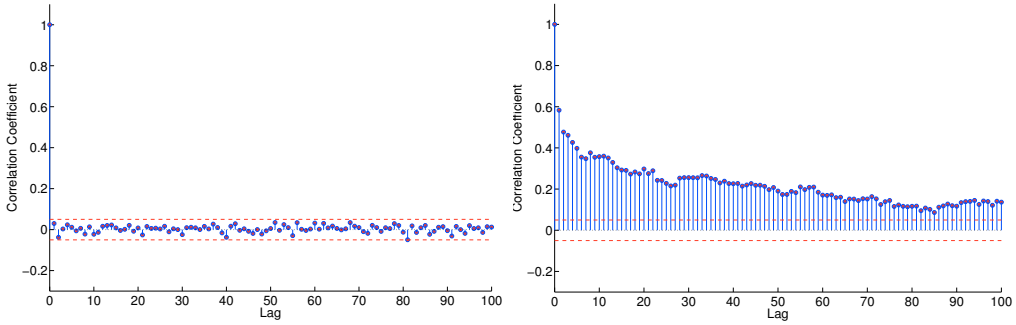


Figure 2.24: PS Autocorrelation: a Witty host (left) and a DNS server (right).

relation of Witty payloads is obviously a consequence of the random padding, whereas the presence of correlation in DNS traffic might be explained by the application-protocol structure and by the content of the DNS reply. It is also interesting, however, that we found a similar distinction as regards packets

IPTs. Those observed from Witty hosts are uncorrelated at all lags (both for CAIDA and MAWI traces), whereas for DNS hosts IPTs are correlated at several lags (Fig. 2.25).

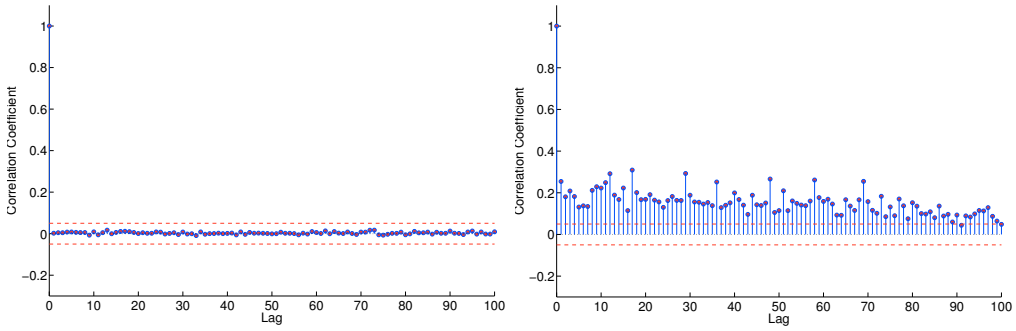


Figure 2.25: IPT Autocorrelation: a Witty host (left) and a DNS server (right).

Analysis of Slammer Traffic. The scarce availability of Slammer traces does not allow to compare worm traffic of two totally different links/observation sites, neither offers the possibility to observe and compare it against large quantities of legitimate traffic flowing in the same links. However, they offer another interesting insight: the 2 MIT links belong to the same backbone and the anonymization algorithm applied to both traffic traces is the same. Therefore we were able to recognize the presence of some infected hosts in both traces, allowing to compare their behavior in both links.

As reported earlier, PS is fixed in Slammer. Therefore, from a packet-level point of view, we limited to the analysis of IPTs. Even from this aspect, Slammer hosts behave more heterogeneously than the Witty ones. The IPT PDFs calculated for the single hosts show more differences, however the average PDF is able to represent them. In all cases the profiles found were quite different from those of DNS traffic shown earlier, and from those of the small portion of legitimate traffic found on the original MIT traces.

In Fig. 2.26 the average PDFs and CDFs of Slammer hosts on both links are compared. The very similar shape of the curves is an interesting invariant. Whereas, the shift of the curves of *Link 1* when compared to those

of *Link 0* is explained by the fact that the first link routes less traffic. This difference is testified by the mean and median values reported in Tab. 2.12, while the other parameters (e.g. entropy) show general consistency.

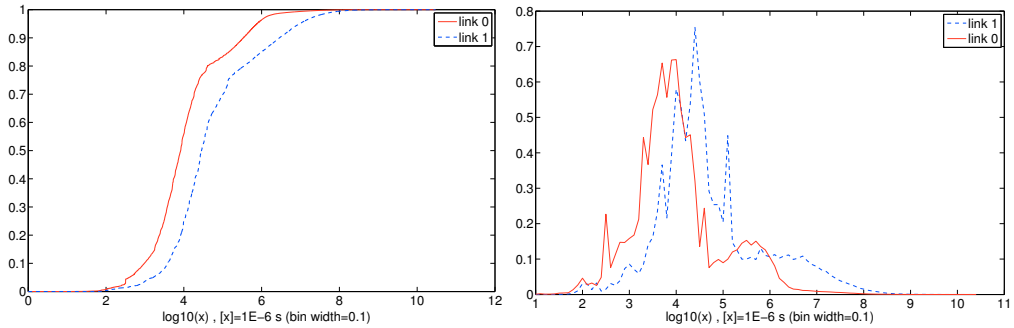


Figure 2.26: CDFs (left) and PDFs (right) of Slammer IPTs.

Table 2.12: Slammer IPT $(log10(x), [x] = 1E-6s)$.

Source	Mean	Median	Max	StdDev	Entropy (bit)
Link 0	4,046	3,918	10,47	0,953	8,00
Link 1	4,710	4,455	10,46	1,123	7,70

Like the results found for Witty (and differently from DNS traffic), IPTs are basically uncorrelated for all the Slammer-infected hosts we analyzed. However most of them present a very small correlation which oscillates around 0 from lag 0 to lag 100. This can be observed from Fig. 2.27, where two different hosts from the two links are analyzed.

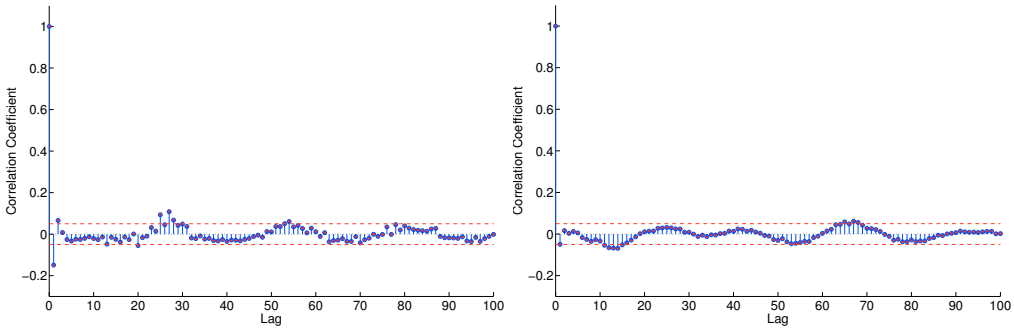


Figure 2.27: IPT autocorrelation of two Slammer hosts, on *Link 0* (left) and *Link 1* (right).

Discussion. This study represents a first step to understand more deeply the impact of worms on network traffic, giving insights which we plan to use for classification and detection purposes. We show that worm traffic presents interesting properties of spatial and temporal invariance, and looks very different from other kinds of traffic. This aspect is reflected by: (i) joint characterization of the marginal distributions of PS and IPT shown by means of joint PDFs; and (ii) lack of temporal (auto-)correlation in the PS and IPT time series. In [48] more results and details regarding the traffic analysis on this category of traffic are reported.

2.3 From Traffic Analysis to Traffic Classification

In this chapter we have shown several examples of how, by looking at network traffic at packet-level, we can gather insights into the properties of different network applications that are useful to better understand them and the traffic they generate.

We started by analyzing traditional Internet applications like HTTP and SMTP. We have shown how each of them presents very distinctive properties that tend to be invariant with respect to time and observed link. In the case of HTTP we also observed how extraneous traffic using the same ports (e.g. Peer-to-Peer file sharing) significantly alters the profiles we found. When analyzing a novel category of traffic, looking at Counter-strike, we noted properties very different from typical Internet traffic (e.g. very small packets) but, again, very regular and independent by the link and the network scenario considered. In the case of another novel category of applications, Peer-to-Peer TV, by representing traffic through joint distributions of PS and IPT, and through scatter plots of average PS and number of packets, we were able to build heuristics to discriminate between the signaling and the data traffic of such applications. At the same time we observed common behaviors of the four applications considered, which belong to the same application

category. Finally, when looking at two different computer worms, again by considering joint distributions of PS and IPT, but also by evaluating the correlation functions, we were able to identify possible fingerprints of the two worms. For example, by comparing traffic from Witty-infected hosts against legitimate DNS traffic we showed that joint distributions present very distinctive profiles both for data averaged on multiple hosts and for single hosts (even when the two traffic compared used the same transport protocol, and were captured on the same link and at the same time). The low correlation of PS and IPT sequences of the two scanning worms, compared to the much more correlated sequences of legitimate traffic, also allowed to spot criteria for the identification and detection of this category of malware through traffic analysis.

It is therefore evident how the distinctive properties here shown, and the techniques for the extraction, analysis, and representation of traffic data that we developed, can be useful in identifying and classifying network applications by looking at the traffic they generate. In Chapter 5 we will indeed present an original technique for traffic classification relying on the extraction of features from the joint distributions of PS and IPT. This technique shows interesting properties of robustness (i.e. difficult to evade) because based on distinctive behaviors of different categories of network applications.

Moreover, we want to point out that the findings highlighted in this chapter are only a part of the knowledge that was useful to us in producing the contributions presented in the next chapters. Another relevant part, indeed, is the background knowledge and the techniques acquired and developed when working in the field of traffic analysis. As will be clear in the following, traffic classification can be partially viewed as a specific sub-field of traffic analysis. It heavily depends on its techniques and tools, it exploits several findings as input to identification techniques, and it takes advantage of the know-how from traffic analysis (and modeling) for the development of classification approaches. In this regard, the software platform for traffic classification, called TIE, that we present in Chapter 4 represents an example

with respect to software tools and techniques. TIE borrows (and extends) many concepts and techniques from Plab, the platform that we developed as an aid in our studies of traffic analysis presented in this chapter: packet capture, as well as the decomposition of traffic into sessions, and the extraction of measurements at different levels (which become *features* when talking about traffic classification).

Starting from the next chapter we define the problem of traffic classification in detail, and in subsequent chapters we then present our contributions in this research area.

Chapter 3

Internet Traffic Classification and Identification

3.1 Introduction

In general, to "classify" network traffic means to analyze traffic in order to ascribe distinct portions of it to the network applications generating them. In both industrial and academical research, interest in this task has dramatically grown in the past few years. This is because the traditional approach to classify traffic by using transport-layer protocol ports is becoming increasingly unreliable, while finding an alternative solution (as shown in the following) has been demonstrated to be a non-immediate task. Network traffic classification is fundamental to build knowledge on the use of network links, but it is also a crucial functionality to impose security or quality-of-service policies, to perform accounting, and many other relevant tasks. In recent years research community and networking industry have investigated and developed several approaches. None of them solves the problem definitively and all of them show some drawbacks related to issues such as on-the-field applicability, reliability, privacy. Moreover, while the state of art is rapidly improving, Internet protocols and applications are continuously evolving (encryption is a notable example closely related to traffic classification), opening new challenges.

This chapter aims at analyzing traffic classification starting from an in-

depth study of all the motivations and difficulties that make it an open topic of research. We provide a critical review of the related literature and, in order to motivate the rest of the work presented in this thesis, we highlight some of the most challenging problems to be faced today by the research community devoted to this field.

3.2 Motivation

Why is traffic classification important? A plausible answer for a researcher would probably be "knowledge". The Internet is a system of enormous complexity subject to continuous evolution. Measurements and analysis of what happens inside this system, for example in terms of network traffic, are therefore of paramount importance to improve our knowledge of it. We stressed this aspect in the introduction of this thesis when talking about traffic measurements and analysis in general, and traffic classification is actually an important part of traffic analysis. If we cannot classify Internet traffic then we have no clue of what our links carry, we do not know how people are using the Internet, what is the new killer application, etc. This is the first step, for example, to understand the impact of new network applications on our links. Is it really relevant to characterize or model the network traffic generated by a specific application or not? Are we doing it under the right assumptions on network conditions, user population, distribution of nodes etc.? Moreover, history has shown that links can suddenly be traversed by traffic related to new forms of malware (e.g. worms, botnets). Being constantly updated on the categories of traffic in our networks and able to rapidly sketch new trends and phenomena is important to be prepared and react to new threats.

On the other side, there are several practical applications of traffic classification, often related to the needs of network providers and operators, that drive the research in this field. Classifying network traffic is for example fundamental to operate a conscious network provisioning. Understanding the which network applications are used (and their quality of service requirements through, e.g., traffic analysis) allows to better allocate resources and

investments. At the same time service providers are looking for new sources of revenue, given the high cost of deploying physical infrastructure and operating the network, and traffic classification seems to play a key role in it. Content-based charging, differentiated accounting and billing depending on network services, the offering of multiple service level agreements customized on categories of applications, are all means for providers to participate in the more profitable value chain of Internet applications [2].

Sometimes papers on traffic classification focus on the detection of specific (kinds of) applications, we usually adopt the term of *traffic identification* for this purpose. This usually happens when such applications are particularly difficult to recognize (e.g. because of encryption or protocol obfuscation) or when there is strong interest in accurately detecting them. Several efforts have been made for example in the detection of Peer-to-Peer file sharing applications [65] [55] and in Skype traffic [66][67] [68]. For instance, in the case of Skype, network operators offering integrated services may be interested in understanding how much their customers use it for voice calls rather than simple chatting.

It is also evident how the ability to ascribe each single flow of traffic to a specific network application is fundamental to perform on-the-fly activities like resource allocation, traffic shaping, service differentiation and quality of service (QoS). Several papers regarding traffic classification show the specific purpose of supporting QoS enforcement [69]. In Chapter 4 (Section 4.10) we report on the integration of the online traffic classification platform that we developed into a QoS framework for new generation networks which is part of an European project funded by the EU.

Finally, traffic classification is very relevant in the field of network security. After being classified, traffic flows can be subject to the enforcement of specific security policies through for example packet filtering performed by routers and firewalls. Several network operators are indeed interested into limiting, regulating, or denying the use of certain services (e.g. Peer-to-Peer file sharing, chat, etc.). Firewalls evolved from using protocol ports

to application-level proxying and payload inspection. However, application-level proxies can be sometimes evaded by protocol encapsulation (e.g. chat or file sharing traffic encapsulated into HTTP), while payload inspection can be tricked by protocol obfuscation techniques and cannot be applied (under common conditions) when traffic encryption is in use. Thus, the search for reliable and robust classification techniques is fundamental for the proper enforcement of security and network-usage policies. Moreover, traffic classification can be useful in detecting malware traffic, for example related to worm spreading or attacks[70].

3.3 What makes traffic classification a difficult task

Traffic classification is today an open problem because of several reasons. As anticipated, initially Internet traffic was classified on the basis of transport-layer protocol ports, e.g. port TCP:80 was assigned to HTTP (which initially was only Web browsing), port TCP:25 to SMTP, and so on. The Internet Assigned Numbers Authority (IANA) [71] assigns the well-known ports from 0-1023 and registers port numbers in the range from 1024-49151 to applications, whereas ports from 49152 through 65535 are defined as "dynamic and/or private". Since several years port-based classification has not been reliable [72] [53] because (i) many applications have no IANA registered ports while they use numbers already registered by others; (ii) many applications use random ports numbers or allow users to define any port number; (iii) often applications are configured to use well-known ports to disguise their traffic and circumvent security and network-usage policy enforcement; (iv) sometimes several servers share a single IP address, thus they need to offer their services through different ports by using network (and port) address translation.

The need to find new approaches to associate traffic flows to applications therefore arises. This is difficult today also because of the proliferation of new

applications running over undisclosed proprietary protocols. Skype [67] and P2P-TV applications [34] are two notable examples. Accurate studies based on traffic analysis and manual payload inspection are usually necessary to understand such protocols and to identify properties useful for classification purposes. In Chapter 2 (Section 2.2.4) we have presented a detailed study of four P2P-TV applications showing how it is possible to derive properties to identify such traffic and to discriminate between their signaling and data traffic. However, new applications emerge continuously and it is difficult to investigate each of them in order to update approaches and/or signatures. Moreover, some applications today use protocol encapsulation, which can confuse several classification techniques, especially those relying on signatures built from payload carried by packets. Examples of such applications are Peer-to-Peer file sharing applications (e.g. KaZaA) [73] and chat applications (e.g. MSN) [74].

Moreover, the use of encryption makes the task of traffic classification even harder. This is straightforward for classification techniques based on payload inspection, which, except for few isolated cases [75], are unable to cope with encrypted data in absence of the decryption keys. However encryption can also alter the normal behavior of traffic patterns, making the work of traffic classification harder even for approaches based on statistical properties of network traffic. Traffic can be encrypted in different modes, because the protocol implements encryption natively (e.g. Skype) or because the traffic is tunnelled into an application-level protocol (e.g. SSH) or in Virtual Private Networks (e.g. through IPSEC); both make traffic classification particularly hard [67] [76], while we may reasonably expect that traffic encryption will become more commonplace as Internet users become more security-savvy.

As explained in Section 3.2, we often need approaches able to work *online* in order to report live information or to take actions according to classification results. The increasing bandwidth of modern network links poses stringent requirements to the speed of classification algorithms and their

computational complexity. Payload-based techniques (explained later) are typically the ones that suffer such problem more than the others. Moreover the large quantity of traffic makes tasks as manual inspection, trace storage, and logistics in general, much harder.

Finally, traffic classification is obviously a hot topic when user privacy is a concern. Available techniques today are evaluated also in terms of how much invading they are with respect to private content carried by traffic. Often, some classification techniques cannot be applied because the access to portions of packets data is not allowed, or some *offline* techniques cannot be experimented because private data cannot be stored.

All the above issues put together make classification of network traffic a challenging task, involving questions related to computational efficiency, social issues, and continuously evolving technical aspects related to new typologies of traffic. In the next sections we give some basic definitions related to traffic classification and summarize the most interesting works produced by the scientific community.

3.4 Definitions

In this section we give some brief definitions in order to better understand the variety of items and topics proposed in literature. A first distinction to be made depends on the classification level. The discrimination can be among (i) traffic classes (e.g. bulk, interactive, ...); (ii) application categories (e.g. chat, streaming, web, mail, file sharing, etc.); (iii) applications (e.g. KaZaa, Edonkey, IMAP, POP, SMTP, ...). Approaches designed for one of such levels of discrimination cannot always work for the others, and their choice usually depends on the purpose behind the proposed classification technique (e.g. class of service mapping). There are also works in literature specifically devoted to identifying the traffic of a single application (e.g. Skype), which usually share similar techniques with the above-mentioned more general approaches, but are adapted to the specific sub-problem.

A second distinction is on the objects to be classified. We often use the

general term *flow*, however, strictly speaking, different kinds of classification objects are considered in literature:

- **TCP connections**. Some works in literature focus only on them. Heuristics or TCP state machines are used to identify the start and the end of each connection.
- **Flows**. The commonly accepted definition of traffic flow is given by the tuple made of $\{source_{IP}, source_{port}, destination_{IP}, destination_{port}, transport\text{-}level\ protocol\}$, plus a flow timeout (values like 60s or 90s are common).
- **Bidirectional Flows** (*biflows*). Same as above, but both directions of traffic are considered.
- **Hosts**. Sometimes, classification approaches want to simply classify the main behavior of a host in terms of the traffic it generates.

Obviously, the techniques presented differ because of the approach used to discriminate different objects and assigning them classes. It is possible to sketch four main categories:

- **Port-based** approaches. These are based on IANA port assignment and on common knowledge of ports typically used by applications that do not have registered ports. A very detailed and constantly updated list is kept at [77].
- **Payload-based** approaches. These techniques inspect the payload content at transport level in order to identify strings, related to the application-level protocol (and in general to the application), matching a set of pre-defined rules.
- **Flow-features-based** approaches. Such techniques are usually based on machine-learning classification techniques applied to features extracted from traffic flows. We generally talk about flow-features but

such features may regard flow-level information and statistics (e.g. flow duration, byte transferred) or even packet-level statistics regarding the specific flow (e.g. vector of packet sizes). However they are based on properties that can be obtained by only looking at transport-level headers.

- **Behavioral and host-based** approaches. They are based on the interactions of the host under observation with the rest of the world, usually in terms of number of connections opened, ports used, and also by using mixes of the above techniques to sketch a typical profile of the host to be compared against profiles previously stored.

With specific regard to techniques using machine-learning approaches. There are many different categories of algorithms that have been used in literature with the purpose to perform traffic classification. In general, machine-learning algorithms can be separated into **supervised learning** and **unsupervised learning** (or clustering). Supervised learning requires training data to be labeled in advance and produces a model that fits the training data. The advantage of these algorithms is that they can be tuned to detect subtle differences and they clearly label the flows upon termination, unlike the unsupervised ones. Unsupervised learning essentially clusters objects with similar characteristics together. The advantage is that it does not require training, and new applications can be classified.

Traffic classification techniques can also be differentiated in ***online*** and ***offline*** techniques. The first ones can be used on-the-field on a network device *listening* to network traffic. They usually have the ability to perform classification of a flow before it ends. Such techniques need to be particularly lightweight or implementable in hardware to keep up with high packet rates of large network links.

Finally, a relevant aspect in the study of traffic classification techniques is how the reference data to evaluate the correctness of classification results is built. This is usually called the ***ground-truth***. These systems usually build upon payload inspection techniques, sometimes supplemented by heuristics

and manual inspection. In literature we have also seen the use of protocol ports for building the ground-truth. This approach is questionable for the aforementioned reasons about unreliability of port-based classification.

3.5 State of the Art in Traffic Classification

The literature of recent years is rich of proposals and evaluations of approaches to traffic classification. Following a temporal order, the association of transport-layer ports to specific applications was, as mentioned, the first classification technique. Even if it performs poorly, e.g., between 50% and 70% accuracy in classifying flows [72] [53], this method is still used today and represents the fastest and simplest one. Port-based classification is used especially when continuous monitoring is in place and there is the need to build online reports. Such technique is supported by several tools. In particular, we cite CoralReef [77] for the ability to process huge quantities of traffic in time, by supporting several advanced network interfaces and frameworks for traffic capture, and to automatically update historical graphical reports for the Web.

When the networking community started to understand that port-based flow identification was becoming unreliable, payload-based approaches started to emerge, especially in the context of network security[78]. Payload-based approaches inspect packets' content to identify byte strings associated to application, or to perform more complicate tasks as syntactical protocol verification and protocol conformance. The most common approaches however are based on a set of signatures (typically in the form of regular expressions [79]) that are checked against packets' content (this is also known as pattern matching). Such approaches are commonly considered very accurate, so that systems to build ground-truth are typically based on them [80, 81, 55]. The main drawback of payload inspection techniques, however, is the computational load required by algorithms performing pattern matching and syntax analysis on contents of network packets. In [82] a useful classification of payload inspection techniques with specific regard to computational aspects

is presented. The computational load required by such algorithms can indeed be prohibitive when we want to perform online traffic classification on a high-bandwidth link. A lot of research is therefore focused on improving algorithms for pattern matching and their implementation in hardware; such interest is high also because the same techniques are used in the field of intrusion detection [83]. Solutions based on programmable hardware and ASIC have been typically proposed in literature and developed by the industry [84].

Another problem of payload-based approaches is privacy. Depending on the usage policy of the network operator and on local laws, access to packets' content might not be possible even by automated software. More often, regulations do not allow to store and archive such data, so that traffic traces to be later processed by payload-inspection techniques cannot be collected. Finally, payload-based approaches can be sometimes circumvented by protocol obfuscation techniques and protocol encapsulation (e.g. applications running on top of HTTP might be erroneously identified as Web traffic), and they are typically ineffective when traffic encryption is in place [85].

Several works in literature have shown the value of payload signatures in traffic classification [86, 55, 87, 72], while others have proposed automated ways to identify such signatures [75, 88, 89]. Most of the approaches used in literature are based on open-source software developed by the networking and research community [80, 81], however there are also several commercial solutions performing protocol identification, some examples are Cisco's NBAR [54] and Juniper's Application Identification [90]. There is usually poor information available regarding how the payload inspection is performed in these proprietary systems.

Because of the several limitations of payload-based techniques, the research community started investigating alternative approaches, both from the point of view of traffic features to exploit for application identification and from the point of view of the classification techniques to be applied to such data. The area of traffic classification using flow-features is the one that probably produced most literature in the field of traffic classification, with a

variegate set of approaches that differ in several of the definitions given in the previous section: level of traffic discrimination, classified objects, classification algorithm, offline/online algorithm design, etc. The flow-features proposed vary very widely, from transport-level header fields to flow-level properties like flow duration, flow size, etc. Features related to the single packets inside flows are also considered, as the vector of the first n packet sizes transmitted, or the mean value of packet sizes inside a flow. In [91], Moore et al. compiled a list of more than 240 flow-features that reflect quite accurately the different kinds of properties considered by the literature in this field. As for the classification algorithms, they span from pattern recognition techniques based on statistical models to data mining and machine-learning algorithms.

In 2004 Wright et al. [92] presented an approach to traffic classification that applies Hidden Markov Model (HMM) techniques to perform multiple sequence alignment. Flows that align close to one another (that have similar subsequences) are viewed as belonging to the same protocol. The authors propose two alternative techniques, one based on the sequences of packet sizes, the second based on packet inter-arrival times. This represents one of the first works in the field of traffic classification through flow-features. Results looked promising even if they were based on traces from 2003, when the novelty and heterogeneity of applications in the link was certainly lower than in the very recent years. They considered a selected set of TCP-based applications pre-classified by looking at port numbers.

In [70] Moore and Zuev propose different variations of supervised machine-learning algorithms based on Naive Bayes. This is one of the few works in literature considering the whole traffic captured from a link without excluding some application categories or some transport-level protocols (e.g. applications running on UDP). The authors use features like flow duration, ports, mean and variance of payload size or inter-arrival times, to classify objects that are unidirectional flows. The classes considered are application categories (e.g. MAIL = IMAP, POP, SMTP). Results show values of accuracy

(i.e. number of correctly classified flows divided to total number of flows) around 95%. However the approach makes use of the protocol ports feature, which in traffic from year 2004 would definitely possess a strong discriminating power on legacy applications. DNS flows are a good example of flows easily recognizible by port number. At the same time, DNS flows typically represent a considerable percentage of the total flows found in a trace, but they usually carry a very small quantity of traffic. Correct classification of such flows makes the accuracy increase, but when the accuracy is evaluated in terms of bytes instead (number of correctly classified flow bytes divided by the total amount of bytes in the trace) it typically decreases. The authors indeed report byte-accuracy values between 79% and 84%. This is an important aspect often underestimated by literature (see next section).

Supervised Naive Bayesian classifiers have been also used in other works (e.g. [93]). Whereas in [94] Zander et al. presented an unsupervised Bayesian classifier (AutoClass) which is tested against traces from 2000 to 2003 using only ports to pre-classify the data with ground truth. The approach classifies bidirectional flows into specific applications (only nine applications are considered). The same AutoClass algorithm evaluated here is then compared by Erman et al. [95] against two unsupervised clustering algorithms that were previously not used for network traffic classification: K-Means and DBSCAN. The results obtained by the AutoClass algorithm are reported as the best ones, however the DBSCAN algorithm shows interesting potential. In this work part of the traces have been pre-classified by only using ports, whereas payload inspection was used for building the ground truth of the other traces, however only few applications were considered from them, with no UDP-based applications at all.

Auld, Moore and Gull (in [96]) proposed an approach with substantial differences with [70]. They made use of a Bayesian Neural Network to perform classification of TCP connections and they removed the protocol ports among the features used. The system proposed achieved very high accuracy (95%) also when it is evaluated under more realistic conditions: the learning

phase is performed with a one-day trace, whereas the testing was done with a trace captured eight month later.

Around year 2006 the first works focused on online traffic classification started to appear. In [97], Bernaille et al. proposed an online approach based on the vector of packet sizes from the first (up to ten) packets exchanged in TCP connections. They evaluated three clustering techniques: K-Means, Gaussian Mixture Model, Spectral Clustering. The authors were able to achieve very good results with very few packets (up to four). However they noted that such technique can be easily circumvented by padding the size of only the first few packets of each connection. A commercial tool [98] was used for ground truth, only TCP connections were considered and no byte-accuracy was reported. In [99] Crotti et al. proposed statistical fingerprints based on packet size (PS), inter-packet times (IPT), and the order of packets. Such fingerprints are basically made of joint PDF of PS-IPT from the *i-th* packets of flows from the same class (e.g. all *second* packets from HTTP connections of the training set will build the joint PDF associated to the *second* packet that is observed for a connection). The authors evaluated the algorithm at the varying of the number of packets (thus of PDF) to be used for online classification. However, only three simple applications were considered as classes: HTTP, SMTP, POP3.

Very recently, in [100], Wei and Moore performed a study focused on the trade-off between selecting good classification features and the latency introduced by collecting and computing them online. They also tested several machine-learning algorithms by using the tool Weka [101] and selected the C4.5 decision tree algorithm because of best performance (99.8%). Again, only TCP traffic was considered and traces from year 2004 are used. TCP connections are the only objects taken into account also in [102], but with traces from year 2006 the realtime implementation of their algorithm was not able to achieve an accuracy greater than 77.5%. A novel contribution of this paper however is that they proposed a *semi-supervised* machine-learning algorithm: first clustering is applied to the data, then available labels from pre-

classification are used to map part of the clusters while other clusters without known labels stay unmapped. This approach, referred as *semi-supervised learning*, has recently received considerable attention in the machine-learning community[103]. In [104] the authors present a detailed survey on traffic classification through machine-learning techniques, by categorizing and qualitatively reviewing several studies in terms of their choice of machine-learning strategies and primary contributions to the traffic classification literature.

Approaches based on the observation of host behavior and of communication patterns, instead, are typically based on heuristics. However some of them have been thoroughly verified and part of these approaches are used to complement payload inspection techniques in ground-truth systems. Some heuristics are indeed particularly useful when classifying peer-to-peer traffic that uses encryption or protocol obfuscation for some flows. For example, keeping a table of *(IP, Port)* pairs for each flow that has been classified by payload inspection, helps to identify unmatched flows that have a source or destination pair stored in the table. Several heuristics have been presented and evaluated in literature [55] [105] [72].

Some approaches are more ambitious and propose novel viewpoints to traffic and representations. In [106] the authors shift the focus from classifying individual flows to associating Internet hosts with applications, and then classifying their flows accordingly. They do this by adopting a multi-level observation of the traffic that comprises looking hosts at the *social level*, by examining the popularity of a host and communities of hosts interacting with each other, while connecting this to their transport-level interactions. For this last point the authors proposed an original approach of representation and fingerprinting of different applications called *graphlets*, based on the observation of simple flow-level information. Similarly, Iliofotou et al. recently proposed Traffic Dispersion Graphs that can potentially be used to classify applications using the network-wide interactions of hosts [107]. The common point of such approaches is that they try to capture interactions observable even with encrypted traffic and are thus targeted to most recent applications

and trends.

Recent literature has seen several works devoted to the identification of traffic from specific applications, because of their impact on overall link traffic and/or because their use of encryption, which makes harder this task. We already cited Peer-to-Peer file sharing applications, for which techniques based on connection patterns and on payload-based signatures have been developed [53] [55] [87] [65]. Skype is also subject of considerable interest: the protocol is not public, different releases of the software behave differently, and the data is encrypted. In [108] the authors propose an identification technique that assumes no access to the payload. The technique involves identifying candidate hosts (and the port they use for Skype traffic) based on properties of special signaling flows used by Skype, and then trying to detect voice calls by using techniques similar to those applied by general flow-features-based approaches. In [66], Suh et al. present a technique to detect relayed Skype flows by comparing input and output traffic patterns. In [67, 68, 109] the authors present techniques to identify Skype traffic based on two subsequent steps: (i) to detect candidate voice flows they first use a Naive Bayesian Classifier applied to a stochastic characterization of Skype traffic in terms of inter-packet gap and packet length; (ii) they then detect a Skype fingerprint from the packet framing structure based on the observation that protocol obfuscation on headers will still keep a recognizable entropy profile in them. This is done using the Pearson's Chi Square test. The last step, however, requires access to packets payload.

More studies are recently targeted to the classification of encrypted traffic in general. In [110], Bernaille et al. extend the classifier presented in [97] to propose a method to detect applications in SSL encrypted connections. Their approach runs in three steps: recognition of SSL connections, detection of the first packet containing application data, and recognition of the encrypted applications by using only the size of the first few application packets (as in [97]) applied to a clustering algorithm based on Gaussian Mixture Models.

Wright et al. [85] use approaches similar to other flow-level classifiers.

Even if, compared to [110], their mechanism requires all packets in the connection (instead of few) before classifying it, it must be noted that their approach is targeted to tunnels of aggregate encrypted traffic (as in VPNs) instead of SSL connections. Their machine-learning approach relies on a k-Nearest Neighbor (k-NN) algorithm applied to packet-level features including packet arrival order. More approaches to classification of encrypted traffic are also presented in [76] and [111].

We conclude this section highlighting that up to today, despite the large quantity of works published in the past few years, no implementations of traffic classifiers that do not rely on protocol ports or payload have been made available. This is in contrast with two facts: (i) scientific papers seem to confirm that it is possible to classify traffic by using properties different from payload; (ii) there are strong motivations for classification in general, and important reasons to perform classification without relying on packet content. We do not have an explanation for this contradiction, we can only make two hypothesis: (i) despite issues related to computational load and traffic encryption, payload inspection is still effective in identifying traffic, and industry is still not significantly investing on alternatives, while privacy concerns seem to be pushed into the background; (ii) effective traffic classification without relying on the mentioned properties is still more difficult to implement than it seems from scientific papers. We believe that research must give more precise answers to questions regarding which, and under which conditions, viable alternatives to payload inspection exist, and this can be done only with the availability of real implementations that must be tested under different situations and with heterogeneous categories of link traffic. We already cited implementations of classifiers relying on ports ([77]) and payload ([81] [80] [54]). As for implementations targeting alternative approaches, NetAI [112] is a tool able to extract a set of features both from live traffic and traffic traces. However it does not directly perform traffic classification, but relies on external tools to use the extracted features for such purpose. To the best of our knowledge the only traffic classifier implement-

Table 3.1: Summary of selected literature on traffic classification

Paper	Full Trace	Year of Traces	Ground Truth	Object
[92] (WRIGHT)	N	2003	Ports	TCP conn.
[70] (MOORE)	Y	$\leq$ 2004	Manual + Payload	Flow
[96] (AULD)	N (TCP)	$\leq$ 2004	Manual + Payload [72]	TCP conn.
[97] (BERNAILLE)	N (TCP)	2003-2005	Payload [98]	TCP conn.
[106] (KARAGIANNIS)	Y	2003-2004	Payload	Hosts/Flows
[114] [94] (ZANDER)	N	2000-2003	Ports	Biflows
[100] (WEI)	N (TCP)	2004	Manual + Payload	TCP conn.
[115] (WILLIAMS)	N	2000-2003	Ports	Biflows
[99] (CROTTI)	N (3 apps.)	$\leq$ 2006	Payload (L7)	Flow
[95] (ERMAN06)	N (TCP)	2001-2006	Ports/Payload	TCP conn.
[102] (ERMAN07)	N (TCP)	2006	Payload (BRO)	TCP conn.

ing a machine-learning technique presented in literature is Tstat 2.0 [113], released at the end of October 2008, which (besides supporting classification through payload inspection) identifies Skype traffic by using the techniques described in [67]. However such techniques have been specifically designed for a single application and can not be extended to classify overall link traffic, and more importantly it partially relies on features built by accessing packet payload. Because of the lack of real implementations and also considering other issues discussed in the next section, we designed a novel traffic classification platform to offer the research community a tool to implement different classification techniques and to compare them. In Chapter 4 we describe our software in detail, showing the main design criteria, the functionalities offered, and the collaborations started with other research groups working on it.

3.6 Open Problems in Network Traffic Classification

Despite the rich literature produced in the field of traffic classification, there are still many open problems that the research community working in this field must address. In this section we summarize what we think are the main topics to be faced in the near future and the problems affecting the current state of art of research in this area.

Techniques. To be concise: we still did not find the *perfect* technique. Firstly, as observed at the end of Section 3.5, there are no prototypes of real traffic classifiers available, unless for those based on payload inspection (and port-based classification obviously). We aim at filling this gap by offering to the scientific community a software platform that will easily allow to develop real implementations of classification technique. Aside, from practical implementations, looking at literature we even do not have a single paper showing an approach applied to whole traffic, showing 100% accuracy both in terms of number of objects and bytes, working online, and without relying on ports or payload. This is obviously a semplification and it is clear that the value of an approach depends also on the context for which it is thought. But, as will briefly argued in the following, the works that have been proposed not only show always some weaknesses, but they are difficult to be fairly evaluated and compared. Table 3.1 reports few of the most notable works on traffic classification and highlights some of the points addressed in this section. In order, each column reports: the paper reference and name of first author, if the entire traffic from a link trace was considered (Y/N) and, if not, which traffic was taken into account, the year of the traces used, the tool/approach used for establishing ground-truth, the objects classified.

Fair Evaluation. Fairly evaluating an approach first requires shared evaluation metrics and working criteria. It seems that we have problems on both of this. As argued in [116] researchers often use disparate metrics in the overall evaluation of classification techniques. The authors highlighted how contrasting definitions of *accuracy* and *True Positives* are often used in top-quality papers. We propose that the networking community adopts common practices from well-established fields, such as the machine-learning and pattern-recognition fields, and agree on few significant metrics shared by everyone.

Moreover, there are metrics specifically needed for the problem of network traffic classification. This is the case of byte-accuracy, which is often

underestimated by literature. In [117] the authors state that byte-accuracy should always be used when evaluating traffic classification algorithms, and that, because of the presence of *mice* and *elephant* flows, neglecting byte-accuracy would lead to what in machine-learning literature is known as a *class imbalance problem* [118]. The authors show that very few of the works in literature report byte accuracy. Besides the class imbalance problem, it is obvious that for most applications of traffic classification (see Section 3.2) it is much more important to correctly classify flows that produce large quantities of traffic. Recently it has also been proposed to use *cost functions* for evaluating classifiers performance depending on the final task for which they have been designed [119]. Another aspect affecting appropriate evaluation is instead related to the data that is processed. First, the final evaluation of an approach - unless highly motivated - should be performed by running the algorithms on the entire traffic that runs on network links. Table 3.1 shows how very few works do this. UDP traffic, for example, is highly neglected, whereas today we know that the percentage of UDP traffic is constantly increasing and new generation applications are often able to work with both TCP and UDP, or sometimes they even *prefer* UDP [109]. Moreover, the traces used are sometimes quite dated compared to the time when the technique is evaluated (see Table 3.1). Because of the motivations behind traffic classification, it is evident that this represents a self-contradiction, and it makes evaluation of the proposed approach doubtful. The same exact objection can be done in the cases we observed that used protocol ports to establish the ground-truth (see Table 3.1).

Comparison and Validation. Comparison among different approaches is the basis for a complete evaluation of the techniques proposed and for a correct understanding of the state of the art. This cannot be done by only looking at metrics shown in papers, because different algorithms need to: be run on the same data, classify similar objects, be tested on more recent traces, use the same reference, etc. However current literature has two main

categories of problems regarding validation and comparison: (i) lack of availability of shared data and implementations; (ii) large differences in several aspects of the proposed approaches. As for availability of actual implementations of traffic classifiers, there are currently very few examples available and most of them are based on payload inspection. If we cannot run them, then we cannot fairly validate and compare them. Moreover, the traffic classification community inherits a historical problem of the traffic measurements and analysis community in general: lack of sharable traces. Often researchers are allowed to work on sensitive data but are not allowed to share it. This problem becomes more difficult in the case of traffic classification, because several algorithms need access to payload data (which raises privacy concerns), and even if we want to focus on techniques that do not need payload, ground-truth creation in general needs to access payload. Two alternative solutions are possible [116] [120]: moving code to the data, moving anonymized data. As for moving the code to the data we need to develop and make available open-source software, in order to make it sharable and observable in its details (the network operator running the software on sensitive data wants to be sure of the operations executed on them). We obviously need collaborative agreements between operators and researchers, and adeguate local laws and regulations. As for moving data, the community is very recently discussing the possibility of making available anonymized traffic traces along with ground-truth obtained before anonymization [121], or alternatively the opportunity of making available anonymous meta-data (features or data allowing feature extraction) along with ground-truth [116]. In the next chapter we present a software platform that we designed to go in the direction of both moving tools to the data (it is open-source and community-oriented) and of distributing of pre-labelled anonymized traffic traces.

The second point afflicting comparability of approaches is their diversity. Table 3.1 shows how they differ in several aspects. We have approaches classifying only TCP connections, other consider flows, others work with bidirectional flows. The tools used for establishing ground-truth differ too, causing

the reference to change, because such tools (as we will show in Chapter 5) are far from being perfect. Moreover, the proposed approaches classify objects into different typologies of classes. E.g., the algorithm in [69] considers only four large traffic classes (Interactive, Bulk data, Streaming, Transactional), other works group applications into categories (MAIL, Web, P2P, etc..) [106] [70], finally most papers consider single applications [99] [97]. Researchers therefore need to agree on creating relations between the different typologies of classes. To perform automated comparison of such algorithms it is necessary to share a well-defined list of applications and of application groups. In the next Chapter we address these issues when designing a software platform for the comparison of classification techniques.

Combination. It is evident that each classification approach offers different advantages. We already pointed out that a perfect technique does not exist. Traffic is variegate and some techniques perform better with respect to some traffic categories or applications than others. For this reason researchers will look in the future at the combination of different classification approaches. This has been already done with respect to heuristics and payload-based signatures. However, the machine-learning community in the last years emphasized the benefit of building multi-classifier machine-learning systems based on the intelligent combination of different algorithms (*experts*) [122]. We have started to see results of applying such approach to the field of intrusion and anomaly detection [123] [124], and we expect to see works in the traffic classification field in the near future. Combination of algorithms obviously requires the definition of common formats even more than what has been advocated earlier in this section. The software architecture presented in Chapter 4 was specifically designed to combine different classification approaches.

Ground Truth: Can We Walk on Solid Ground? First we observe (see Table 3.1) that many works used protocol ports as ground-truth. We already observed that we consider such approach inappropriate. We also

observed how using different techniques can make results more uncertain. Moreover some ground-truth tools are not publicly available and the techniques used are not always well-documented. In Chapter 5 Section 5.5 we show how two of the most common tools do not always assign flows to the same class and how they sometimes are unable to label a non-negligible portion of a very recent traffic trace. This is explainable with the increase of traffic complexity, of the use of encryption, protocol obfuscation and encapsulation. Moreover such tools need to constantly be updated. The research community should therefore: (i) be more aware on the reliability of such tools, (ii) share common techniques and procedures, (iii) work to develop robust and updated techniques. As for the last point, multi-classifier strategies will also help.

Online Classification. We partially discussed this subject in the related work section. The final application of traffic classification in most cases is online classification. Therefore researchers started to produce novel approaches targeted to this direction. Difficulties in evaluating and comparing such systems with appropriate metrics increase because performance issues arise, whereas the lack of availability of real implementations makes comparison harder. While we foresee an increasing effort of the research in the field of online traffic classification, we think that a hot topic in this area will probably be the robustness of such approaches to evasion and obfuscation techniques. The reduced set of features available to online classifiers makes them indeed more vulnerable to such "attacks". In Chapter 5 Section 5.2 we present a new set of features promising interesting properties of robustens to these attacks (but that still need be adapted to be used online). Moreover, the traffic classification platform presented in the next chapter is able to run online, and in Chapter 5 Section 5.4 we experimentally evaluate a classification plugin for such platform targeted to online classification.

3.7 Conclusions

With respect to the scenarios, the problems, and the techniques illustrated in this chapter, and in particular by considering the issues pointed out in the last section, in the next two chapters we present our contributions to the field of traffic classification. In the next chapter, we present *TIE*, a software platform for traffic classification which was designed taking in mind the points highlighted in Section 3.6: TIE is an open-source project, strongly community-oriented, and focused on comparison of multiple techniques, multi-classification, and online traffic classification. In Chapter 5 we present a novel technique to perform traffic classification based on packet-level traffic features, we study the subject of payload inspection by also proposing a novel approach, and we tackle the subject of reliability of ground-truth tools by showing the need for better reference tools.

Chapter 4

A Community-Oriented Platform for Traffic Classification

In this chapter we introduce a novel software tool for traffic classification called *Traffic Identification Engine* (TIE). TIE was designed as a community-oriented tool, inspired by the observations and recommendations made in the previous chapter regarding the current challenges that research in traffic classification must face. Our aim is to offer a common tool for fair evaluation and comparison of traffic classification techniques and to foster the sharing of common implementations and data. Moreover, TIE is thought as a multi-classifier system, supporting the combination of more classification plugins, and its architecture is designed to allow online traffic classification. Other properties that in our opinion make it suitable for extensive use and development by the research community are: public availability of the source code, a web site with rich documentation, well-defined data formats, a developers API, availability of basic classification techniques, and re-use of state-of-the-art definitions, formats, and techniques.

In the following sections we illustrate the main architecture of the software, we give some definitions, and explain main functionalities. Two basic classification techniques implemented as TIE plugins are also described: port-based classification and payload inspection through pattern matching.

We conclude the chapter by describing the involvement of TIE in several international collaborative projects. In Chapter 5 instead, we will present our contributions to the field of traffic classification that have been carried out also thanks to TIE.

4.1 Architecture Overview

TIE is written in C language and runs on Unix operating systems, currently supporting Linux and FreeBSD platforms. The software is made of a single executable and a series of plugins that are dynamically loaded at run time. A collection of utilities and scripts are distributed with the sources and are part of the TIE framework.

TIE is made of several components, each of them responsible for a specific task. Using a modular architecture allows to easily extend, substitute or add a component without modifying the others and also makes code maintenance and development easier.

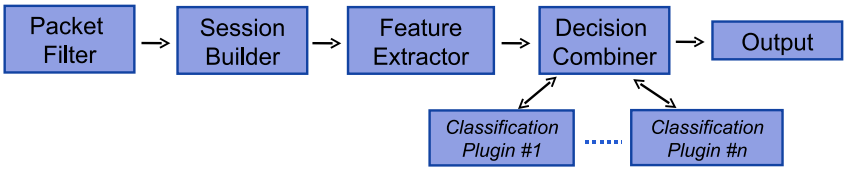


Figure 4.1: TIE: main components involved in classification

Figure 4.1 shows the main blocks composing TIE. The *Packet Filter* is able to both capture live traffic or read from a traffic trace, and it can filter packets depending on several criteria. Packets are then aggregated into separate sessions (as explained in the following, these can be flows, biflows, etc.) by the *Session Builder*, which keeps updated the status of each session. A set of feature extraction routines (e.g. updating statistics on inter-packet times) are performed by the *Feature Extractor*. The classification is performed by the *Decision Combiner*, which coordinates the activities of several classification plugins (each one executing a different classification technique). The *Output* generates final output files with modalities and in data formats

that depend on the operating mode (explained in the following). In the next sections we describe in detail each component and related tasks.

4.2 Operating modes

To implement the very different techniques and approaches existing in literature, TIE supports operation on various kinds of data and different operating modes. In this section we briefly introduce the different operating modes. Their operation will be further defined in the next sections. By default TIE works as a classifier, however it is possible to run it with the only purpose to train one or more classification plugins (e.g. plugins implementing machine-learning techniques). We call it the *training phase*, and we will refer to it in the next sections. However, unless when explicitly stated, in the following we refer to TIE working to perform classification.

To cover different needs of operators and researchers, TIE can operate in three operating modes:

- **Offline**: information regarding the classification of a session is generated only when the session ends or at the end of TIE execution. This operating mode is typically used by researchers evaluating classification techniques, when there are no timing constraints regarding classification output and the user is interested in obtaining information regarding the entire session lifetime. This operating mode can be applied to both live traffic and traffic traces.

- **Realtime**: information regarding the classification of a session is generated as soon as it is available. This operating mode implements *online* classification. The typical application is policy enforcement of classified traffic (QoS, Admission Control, Billing, Firewalling, etc.). Strict timing and memory constraints are assumed. This operating mode should be applied to live traffic, but it can be used on traffic traces for debugging purposes.

- **Cyclic**: information regarding the classification is generated at regular intervals (e.g. each 5 minutes) and stored into separate output files. Each output file contains only data from the sessions that generated traffic during the corresponding interval. An example usage is to build live traffic reporting graphs and web pages. This working mode can be applied only to live traffic.

Obviously the *realtime* mode is the one imposing most constraints to the design of TIE's components. We highlight that TIE was designed since the beginning targeting online classification and this affected several aspects of the architecture that will be described in the next sections.

4.3 Packet Collection and Filtering

As regards packet capture, TIE is based on the Libpcap library[125], which is an open source C library offering an interface for capturing link-layer frames over a wide range of system architectures. It provides a high-level common Application Programming Interface to the different packet capture frameworks of various operating systems. The offered abstraction layer allows programmers to rapidly develop highly portable applications. Moreover it defines a common standard format for files in which captured frames are stored, also known as *tcpdump* format, which is currently a largely used *de facto* standard.

Modern kernel-level capture frameworks on Unix operating systems are mostly based on the BSD (or Berkeley) Packet Filter (BPF) [5]. The BPF is a software device that "taps" network interfaces, copying packets into kernel buffers and filtering out unwanted packets directly in the interrupt context. Definitions of packets to be filtered can be written in a simple human readable format using boolean operators and be compiled in a pseudo-code to be passed to the BPF device driver through a system call. The pseudo-code is interpreted by the BPF Pseudo-Machine, a lightweight, high-performance, state machine specifically designed for packet filtering. Libpcap allows pro-

grammers to write applications that transparently support a rich set of constructs to build detailed filtering expressions for most network protocols. By few Libpcap calls these boolean expressions can be read directly from user's commandline, compiled in pseudo-code and passed to the Berkeley Packet Filter. Figure 4.2 illustrates how TIE, the Libpcap, and the BPF interact and how network packet data traverse several layers to finally be processed and transformed in capture files or in samples for statistical analysis. Finally, Libpcap allows to read packets from files in *tcpdump* format rather than from network interfaces without modifications to the application's code except for a different function call at initialization time. This allows to easily write a single application which can work both in realtime and offline conditions.

To analyze packets, it is necessary to read and interpret link-layer, IP and transport layer protocol headers. Also, it is necessary to get timestamps of when packets were first seen on the interface. Those operations are easily accomplished because of the format of data returned by the Libpcap library for each captured packet. Each time *pcap_next()* returns with success it supplies a pointer to a *pcap_pkthdr* structure and a *u_char* pointer to a contiguous region of memory containing the captured portion of packet. The *pcap_pkthdr()*

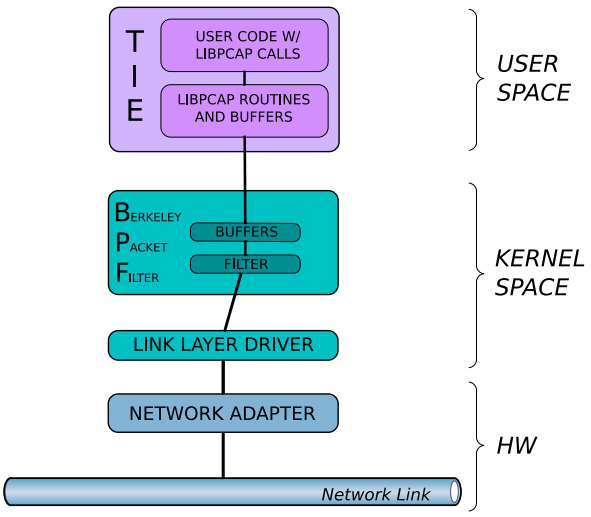


Figure 4.2: Flow of packet data from hardware to the application

contains: the packet timestamp (in microseconds) assigned by the driver, the length of the portion of packet actually captured, the length of the packet as appeared directly on the wire. The timestamp information is fundamental for the calculation of inter-packet times and to evaluate session timeouts (Section 4.4), whereas the pointer to packet data allows to treat the packet as an array of bytes. We defined few elementary macros operating on such array to decode protocol fields. Few examples are reported in Figure 4.3.

```
[...]

/*
 * Headers
 */
/* IP header length in bytes */
#define PKT_IP_HLEN_B(a) ((a[0] & 0x0f) << 2)
/* IP total length in bytes */
#define PKT_IP_TLEN_B(a) ((a[2] << 8) + a[3])
/* TCP header length in bytes */
#define PKT_TCP_HLEN_B(a) ((a[PKT_IP_HLEN_B(a)+14] & 0xf0) >> 2)
/* UDP header length */
#define PKT_UDP_HLEN_B 8

/*
 * Payloads
 */
/* IP payload - valid only for non-fragments */
#define PKT_IP_PAYLOAD_B(a) (PKT_IP_TLEN_B(a) - PKT_IP_HLEN_B(a))
/* IP payload - valid only if applied to last fragment */
#define PKT_F_IP_PAYLOAD_B(a) (PKT_FRAG_OFFSET_B(a) + PKT_IP_TLEN_B(a))
/* TCP payload - valid only for non-fragments */
#define PKT_TCP_PAYLOAD_B(a) (PKT_IP_TLEN_B(a) - PKT_IP_HLEN_B(a) - PKT_TCP_HLEN_B(a))
/* UDP payload */
#define PKT_UDP_PAYLOAD_B(a) (PKT_IP_TLEN_B(a) - PKT_IP_HLEN_B(a) - PKT_UDP_HLEN_B)

/*
 * Ports
 */
#define PKT_SRC_PRT(a) ((a[20] << 8) + a[21])
#define PKT_DST_PRT(a) ((a[22] << 8) + a[23])

/* TCP Flags */
#define PKT_TCP_FLAG_FIN(a) (a[33] & 0x1)
#define PKT_TCP_FLAG_SYN(a) (a[33] & 0x2)
#define PKT_TCP_FLAG_RST(a) (a[33] & 0x4)
#define PKT_TCP_FLAG_PSH(a) (a[33] & 0x8)

[...]
```

Figure 4.3: TIE: Some of the macros implemented to decode protocol fields

As regards packet filtering, we already explained that Libpcap supports the powerful BPF filters, which are called inside the capture driver. More-

over, there are additional filtering functionalities working in user-space that we implemented in TIE. Examples are: skipping the firs m packets, stopping the analysis after n packets, selecting traffic within a specified time range, checking for headers integrity (TCP checksum, valid fields etc.).

4.4 Sessions

TIE decomposes network traffic into sessions, which are the objects to be classified. As seen in Chapter 3 (e.g. Table 3.1), in literature have been presented approaches to classify different kinds of traffic objects: from flows to TCP connections and even hosts. To make TIE support multiple approaches and techniques, we have defined the general concept of session, and specified different definitions of it (selected using command line switches):

- **flow**: defined by the tuple $\{source_{IP}, source_{port}, destination_{IP}, destination_{port},$ *transport-level protocol*$\}$ and an inactivity timeout, with a default value of 60 seconds.
- **biflow**: defined by the tuple $\{source_{IP}, source_{port}, destination_{IP}, destination_{port},$ *transport-level protocol*$\}$, where source and destination can be swapped, and the inactivity timeout is referred to packets in any direction (default value is 60 seconds).
- **host**: a host session contains all packets it generates or receives. A timeout can be optionally set.

When the transport protocol is TCP, biflows typically approximate TCP connections. However no checks on connection handshake or termination are made, nor packet retransmissions are considered. This very simple heuristic has been adopted on purpose, because it is computationally light and therefore appropriate for online classification. In the following we show that this definition simply requires a lookup on a hash table for each packet, as flows (which indeed are used in online monitoring also for the lightweight processing they require). However, some approaches may require stricter rules to

recognize TCP connections, at least able to identify the start and end of the connections with more accuracy. This, for example, because they rely on features based on the very first packets (as TCP options, or packet sizes) [100] [97]. Moreover, explicitly detecting the expiration of a TCP connection avoids its segmentation in several biflows when there are long periods of silence. This behavior is typical for interactive applications like Telnet and SSH.

For these reasons, we implemented also additional heuristics, which can be optionally activated, to follow the state of TCP connections by looking at TCP flags:

- if the first packet of a TCP biflow does not contain a SYN flag then it is skipped. This is especially useful to filter out connections initiated before traffic capture was started.
- The creation of a new biflow is forced if a TCP packet containing only a SYN flag is received (i.e. if a TCP biflow with the same tuple was active then it is forced to expire and a new biflow is started).
- A biflow is forced to expire if a FIN flag has been detected in both directions.
- The inactivity timeout is disabled on TCP biflows (they expire only if FIN flags are detected).

These heuristics have been chosen in order to trade-off between computational complexity and accuracy, keeping in mind the TIE's vocation to work in *online* mode. Some applications, however, may require a more faithful reconstruction of TCP connections. For example payload inspection techniques used for security purposes, may require the correct reassembly of TCP streams in order to not be vulnerable to evasion techniques [126]. For these situations, a user-space TCP reassembly state machine may be adopted and integrated into TIE, however this would significantly increase computational

complexity compromising *online* mode under some circumstances (depending on computational power and link load).

Both *biflow* and *host* session types contain traffic flowing in two opposite directions, that we call *upstream* and *downstream*. Information regarding the two directions must be kept separately, for example to allow extraction of features (e.g. IPT, packet count, etc.) related to a single direction. Therefore, within each session with bidirectional traffic, counters and state information are also kept for each direction. For both biflow and host session types, upstream and downstream are defined by looking at the direction of the first packet (upstream direction).

In order to keep track of sessions status according to the above definitions we use a data structure in which each session can be dynamically stored.

Each session type is identified by a key of a fixed number of bits. For *flow* and *biflow* session types the key contains two IP addresses, two port numbers and the protocol type. For the *host* session type, the key contains only one IP address. An IPv4 address is constituted of 32 bits, a port number is represented with 16 bits and the protocol type requires only 8 bits. So the maximum theoretical number of sessions that can be encountered on a link is 2^{104} working with flows and biflows and 2^{32} working with hosts. Therefore, a static data structure, which would have yield the benefit of a $O(1)$ complexity for data insertion and search, is not feasible in both cases. Complexity for data lookup is very important in the design of a classifier, because an access must be made to this data structure for each single packet. Among dynamic data structures, balanced binary trees offer a worst-case complexity which is logarithmic with respect to the number of elements ($log_2(n)$), but we found that an implementation with hash tables could yield comparable results in all realistic situations. Specifically, sessions are stored in a *chained hash table*. A hash table is basically made of a direct address table (a static array) which is addressed by an index obtained through an *hash function* applied to the original key of the element. Each position into the array is also named *slot*. A hash function performs a m to k mapping with $k < m$, k equal to the length

of the array, also called the hash table size, and m equal to the number of values that elements' keys can assume (2^{104} or 2^{32} in our example). When multiple keys map onto the same integer we say that there is a *collision*. A hash function must be computationally fast and must be designed to reduce the number of collisions as much as possible by fairly exploiting all the slots in the table. In chained hash tables collisions are handled by chaining colliding elements into a linked list. This allows an unlimited number of collisions to be handled and does not require a priori knowledge of how many elements are contained in the collection. When the hash function is well-designed, and the number of slots is larger than the number of elements, collisions are rare and the average lookup and insertion time is $O(1)$. But even when the length k of the array is smaller than the number of elements, if the hash function is designed to generate indexes with uniform probability when applied to the population of the keys, then we can assume an average worst case complexity[1] of $O(n/k)$. This means that, for example, even for $n = 200$ millions of sessions we could choose $k = 10$ millions, obtaining a complexity smaller than $log_2(2e + 08) \approx 27.5$ by allocating an array of pointers filling less than 40 MB of memory.

There are several strategies for maximizing the uniformity of the hash function and thereby maximizing the efficiency of the hash table. One method, called the division method, operates by dividing a data item's key value by the total size of the hash table and using the remainder of the division as the hash function return value. Selecting an appropriate hash table size is an important element in determining the efficiency of the division method. A good rule of thumb in selecting the hash table size for use with a division method hash function is to pick a prime number that is not close to any power of two. Figure 4.4 shows the simple hash function used for biflows. The function has been written so that source and destination hosts' IP addresses/ports can be swapped and still generate the same key.

[1]That is, the average length of the linked lists is n/k

```
        /* source ip */
        for (i = 12, j = 0; i != 16; i++) {
                j = (j * 13) + packet[i];
        }
        /* source port */
        for (i = 20; i != 22; i++) {
                j = (j * 13) + packet[i];
        }

        /* dest ip */
        for (i = 16, k = 0; i != 20; i++) {
                k = (k * 13) + packet[i];
        }
        /* dest port */
        for (i = 22; i != 24; i++) {
                k = (k * 13) + packet[i];
        }

        return ((j + k + L4_PROTO(packet)) % BIFLOW_TABLE_SIZE);
```

Figure 4.4: TIE: hash function used to identify and store biflow sessions.

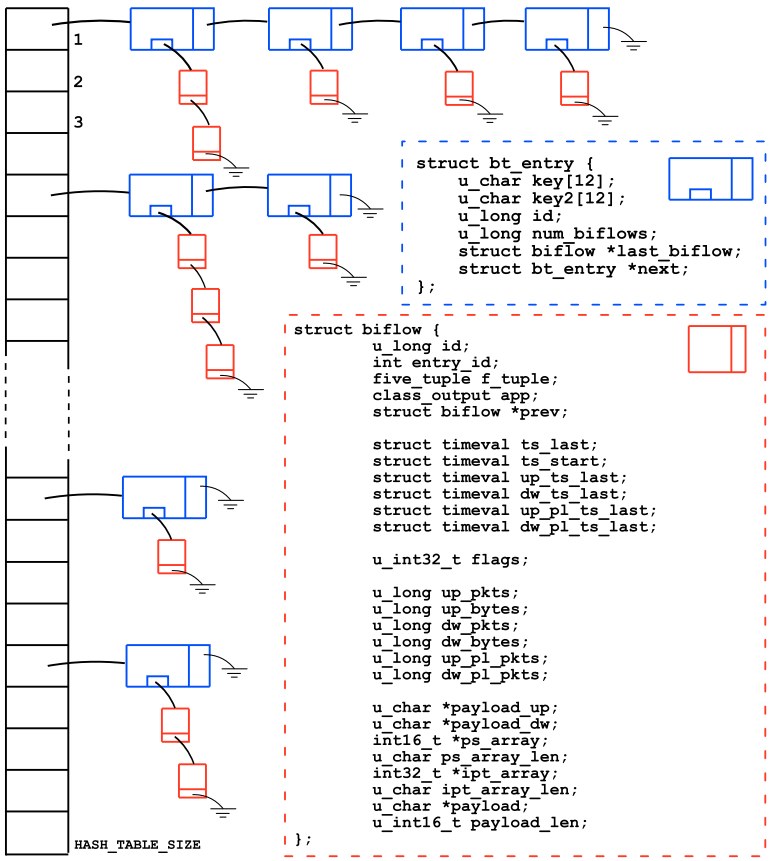


Figure 4.5: TIE: sessions stored into the chained hash table (biflows).

For each session it is necessary to keep track of some information and to update them whenever a new packet belonging to the same session is processed (e.g. status, counters, features). Also, it is necessary to archive an expired session and to allocate a new structure for a new session. We therefore associate to each item stored in the hash table a linked list of sessions structures. That is, each element of the hash table, which represents a session key, contains a pointer to a linked list of session structures, with the head associated to the currently active session. Figure 4.5 represents how sessions are stored into the hash table, and also shows detailed structures related to biflows. These are commented in the next section, related to feature extraction.

In order to properly work with high volumes of traffic, TIE is also equipped with a Garbage Collector component that is responsible of keeping clean the session table. At regular intervals it scans the table looking for expired sessions. If necessary it dumps expired sessions data (including classification results) to the output files and it then frees the memory associated to those sessions. The cleaning interval occurs by default every 10000 packets, but such value can be changed using a command-line option. Working in offline mode the Garbage Collector is responsible of appending classification results to the output file. In cyclic mode its work is synchronized with the dumping process made at regular intervals. Under realtime mode instead, it is only responsible to free memory of expired sessions.

4.5 Feature Extraction

In order to classify sessions, TIE has to collect the features needed by the specific classification plugins activated. For instance, a technique may need to access the payload of the first packet of a session in order to operate a pattern matching using some signatures. The Feature Extractor is the component in charge of collecting classification features and it is triggered by the Session Builder for every incoming packet. To avoid unnecessary computations and memory occupation, most features can be collected on-demand by specifying

command line options. This is particularly relevant when we want to perform online classification. The calculation of features is indeed a critical element affecting the computational load of a classifier. In [100] the computational complexity and memory overhead of some features in the context of online classification are indeed evaluated.

We started implementing basic features used by most classifiers, considering techniques of different categories: post-based, flow-based, payload inspection. At the moment these are the features available to classifiers:

- **Always available**
  - number of upstream/downstream packets
  - payload upstream/downstrem bytes
  - source/destination port
  - transport layer protocol
- **On-demand**
  - Inter Packet Time between the first n packets
  - Packet Size of the first n packets
  - First n bytes of first packets (in both directions in biflow mode)
  - Session payload stream of n bytes

We plan to enlarge the list of supported features by considering both new kinds of features that we illustrate in Chapter 5 and sets explicitly compiled and published in literature [91].

Classification features extracted from each session are kept in the same session structure described in Section 4.4. In Figure 4.5 the biflow structure is shown. In general, each session structure in the table contains:

- **Basic information**: a table entry identifier, a session identifier, the tuple representing the key of the entry (features like protocol ports are part of this tuple), partial or final classification results, and a pointer to a previously expired session.

- **Timing information**: the timestamps of the last seen upstream and downstream packets are used to compute features like inter-packet times [96]. Those related only to packets carrying payload are used to compute inter-packet times when ignoring non-payload packets. The timestamp of the last seen packet, independent of traffic direction, is used to check for session timeout and, together with the timestamp of the first packet, allow us to compute features as the session duration (T_{ON}) and inter session time (T_{OFF}).

- **Flags**: a 32 bit register containing flags about the status of the session. These flags are mostly used by classification plugins (see section 4.6) to properly work on sessions.

- **Counters**: some counters used for features like the number of: packets, packets with payload, and bytes observed (separately calculated for both directions).

- **Features**: Some other classification features that can be enabled through command line options. The first bytes of the first packet, in both directions, are provided for lightweight packet inspection techniques. Payload sizes and inter-packet times vectors are provided for statistical or machine learning techniques[99] [97]. A payload stream vector is finally provided for deep payload inspection techniques [80].

This structure can be easily extended to collect additional features. Moreover the collection of each on-demand feature is implemented as an inline function which can be also enabled/disabled at compile time. For each feature there is a state condition that can be tested by a classification plugin to verify if the required feature is available.

4.6 Classification

TIE provides a multi-decisional engine made of a Decision Combiner and one or more Classification Plugins (or shortly classifiers) implementing different

classification techniques. Each Classification Plugin is a standalone dynamically loadable software module. At runtime, a *Plugin Manager* is responsible of searching and loading classification plugins according to a configuration file called *enabled_plugins*.

```
typedef struct classifier {
        int (*disable) ();
        int (*enable) ();
        int (*load_signatures) (char *);
        int (*train) (char *);
        class_output *(*classify_session) (void *session);
        int (*dump_statistics) (FILE *);
        bool (*is_session_classifiable) (void *session);
        int (*session_sign) (void *session, void *packet);

        char *name;     /* string representing the name of the classification engine */
        char *version;  /* string representing the version of the engine */
        u_int32_t *flags;
} classifier;
```

Figure 4.6: TIE: interface of classification plugins.

Classification plugins have a standard interface, shown in Figure 4.6. To help plugin developers, a *dummy* plugin with detailed internal documentation is distributed as part of the TIE package. Moreover the other classification plugins distributed with TIE (e.g. the Port-based classifier) can serve as sample reference code.

Each plugin is identified by a name and a version number. After loading a plugin, the Plugin Manager calls the corresponding *enable*() function, which is in charge of verifying if all the features needed are available (some features are enabled by command line options). If some features are missing, then the plugin is disabled by calling the *disable*() function. After enabling a plugin, the *load_signatures*() function is called in order to load classification fingerprints. If the loading process encounters an error then the plugin disables itself.

4.6.1 Decision Combiner

The Decision Combiner is responsible for the classification of the sessions and it implements the strategy used for the combination of multiple classi-

fiers. Whenever a new packet associated to an unclassified session arrives, after updating session status information and extracting features, TIE calls the Decision Combiner. For each session, the Decision Combiner must make four choices: if a classification attempt is to be made, when (and if) each classifier must be invoked (possibly multiple times), when the final classification decision is taken, how to combine the classification outputs from the classification plugins into the final decision. To take these decisions and to coordinate the activity of multiple classifiers, the Decision Combiner operates on a set of session flags (Figure 4.7) and can invoke, for each classification plugin, two functions in the *classifier* structure: *is_session_classifiable()* and *classify_session()*. The $is\_session\_classifiable()$ function asks the classifier if enough information is available for it to attempt a classification of the current session. The $classify\_session()$ function performs the actual classification attempt, returning the result in a *class_output* structure, shown in Figure 4.8.

```
/* Session Flags */
#define SESS_SKIP               0x1     /* 1 => skip session processing */
#define SESS_PL_UP              0x2     /* 1 => 1st upstream pkt with payload received */
#define SESS_PL_DW              0x4     /* 1 => 1st dwstream pkt with payload received */
#define SESS_NO_ALPHA           0x8     /* 1 => payload is not alpha-numeric */
#define SESS_DW_START           0x10    /* 1 => 1st pkt with payload was in dwstream */
#define SESS_LAST_PKT           0x20    /* Last pkt direction:0 => upstream ,1 => downstream */
#define SESS_DONT_CLASSIFY      0x40    /* 1 => session is not to be classified */
#define SESS_CLASSIFIED         0x80    /* 1 => session has been classified */
#define SESS_SIGNED             0x100   /* 1 => signature saved */
#define SESS_EXPIRED            0x200   /* 1 => session expired */
#define SESS_TCP_SYN            0x400   /* 1 => SYN flag seen for this session */
#define SESS_TCP_FIN_UP         0x800   /* 1 => FIN flag seen in upstream for this session */
#define SESS_TCP_FIN_DW         0x1000  /* 1 => FIN flag seen in dwstream for this session */
```

Figure 4.7: TIE Session Flags: these are set by the Session Builder, the Feature Extractor, and the Decision Combiner.

To highlight the central role of the Decision Combiner (DC in the following), and how few functions and structures allow a flexible design of its operating strategy, in the following we illustrate some sample situations regarding the four main decision mentioned above.

- **When to attempt classification**. The DC could decide to not evaluate the current session depending both on information from the classifi-

cation plugins or on a priori basis. The latter may happen, for example, when the target of classification is a restricted set of traffic categories. Such decision can be taken by looking, for example, at the session flags or at session features. In the first case instead, the DC typically asks each of the active classification plugins if it is able to attempt classification on the current session (through the *is_session_classifiable()* function. Depending on the replies from the classifiers the DC can decide to make a classification attempt. For example, the DC may wait for all classifiers (or the majority of them) to be ready before making an attempt.

- **When each classifier must be invoked**. Depending on the classifiers that are available, the DC could decide to invoke only some of them, and only at some time, for a certain session. For example, there could be classification techniques that are applicable only to TCP biflows. Or some classifiers may be invoked only when certain information is present. This is the case of payload-based classifiers. In general, we can design Combination strategies with more complicate algorithms, in which the invocation of a specific classifier depends on several conditions and on the output of other classifiers. For example, only if a certain classifier returns that the session could be a Peer-to-Peer session, then a payload inspection technique is launched. Or if the session is classified as encrypted the DC may start a specific classifier specif-

```
/* Generic output given by a classifier */
typedef struct class_output {
        u_int16_t id;           /* Application identifier */
        u_int8_t subid;         /* Application sub id */
        u_int8_t confidence;    /* Confidence associated with match */
        u_int32_t flags;
} class_output;

/* Classification output flags */
#define CLASS_OUT_ERR    1  /* classification error */
#define CLASS_OUT_REDO   2  /* classifier wants to re-examine session
                               when new data is available */
#define CLASS_OUT_NOMORE 4  /* classifier will not re-examine this session */
```

Figure 4.8: TIE: the class_output structure stores the output of a classification attempt.

ically designed for encrypted of traffic. Basically the algorithm which determines the choice and the sequence of the classifiers to be invoked can be very simple or more complex depending on the nature of the classification problem and available classification techniques.

- **When the final classification decision is taken**. This choice is usually connected to the previous one. The DC must decide when TIE has to assign a class to a session. This can happen at the arrival of any packet from the considered session. Simple strategies are, e.g., when at least one classifier has returned a result, or when all the classifiers have returned a classification result, etc. In more complicate approaches, this choice can vary depending on the features of the session (e.g. TCP, UDP, number of packets, etc.) and the output of the classifiers. Moreover, if working in *online* mode, a limit on the time elapsed or the number of packets seen since the start of the session is typically given. If such limit has been passed, a final classification result (even if labeled as *Unknown* or no classifier has ever been launched for the considered session) is assigned.

- **How to combine the classification outputs from the classification plugins into the final decision**. The DC receives a *class_output* structure (Figure 4.8) from each of the classification plugins invoked. These must then be *fused* into a single final decision. The *class_output* structure contains also a confidence value returned by each of the classifiers, which can be helpful when combining conflicting results from different classifiers, and it determines the final confidence value returned by the DC. The criteria used by each classification plugin to assign a value to the confidence value is defined by the designer of the classification plugin and must be clearly reported in the plugin documenation, unless it is always set to the maximum (default). Effectively combining conflicting results from different classifiers is a crucial task. The problem of combining classifiers actually represents a research area in

the machine-learning field *per se*. Simple static approaches are based on majority and/or priority criteria, whereas more complex strategies can be adopted that take into account the nature of the classifiers and their per-class metrics like accuracy [122].

We distribute TIE with a basic combination strategy as a first sample implementation. For each session, the decision is taken only if all the classifiers that are enabled are ready to classify it. To take its decision the combiner assigns priorities to classifiers in the order of their appearance in the *enabled_plugins* file. If all the plugins agree on the result, or some of them classify the session as *Unknown*, the combination is straightforward and the final confidence value is computed as the sum of each confidence value divided by the number of enabled plugins. Instead, if one or more plugins disagree, the class is decided by the plugin with highest priority. To take into account the conflicting results of the classifiers, the confidence value is evaluated as before, and then divided by 2.

All the code implementing the decision combiner is in separate source files that can be easily modified and extended to write a new combination strategy. After future addition of further classification plugins, we plan to add combination strategies that are more sophisticated. For example, in Section 5.5 of Chatpter 5 we suggest the combination of multiple techniques for the creation of a ground-truth system, including payload inspection and heuristics for Peer-to-Peer traffic.

4.6.2 Training Phase

It is possible to run TIE with the purpose to train one or more classification plugins that implement machine-learning techniques with data extracted from a traffic trace. To do this, we first need pre-classified data (ground truth). These can be obtained by running TIE on the same traffic trace using a ground-truth classification plugin (e.g. the l7-filter classification plugin illustrated in Sectionsec:tie:l7). The same output file generated by TIE is then used as pre-classified data and given as input to TIE configured to

perform a training phase. For each activated classification plugin, the *session_sign* and *train()* functions are used The *session_sign* function is called for each packet belonging to a session until the SESS_SIGNED flag is set and should store information needed by the training process into a dedicated data structure managed by the plugin itself. The second one is called at the end of TIE execution and actually performs the training process using previously stored data.

4.7 Data definitions and Output format

One of the design goals of TIE, was to allow comparison of multiple approaches. For this purpose a unified representation of classification output is needed. More precisely (also with the purpose to make classification plugins "speak the same language") we defined IDs for application classes (we simply call them *applications*) and propose such IDs as a reference. Moreover, as observed in Chapter 3, several approaches presented in literature classify sessions into classes that are categories grouping applications that offer similar services. We therefore added definitions of *group* classes and assigned each application to a group. This allows to compare a classification technique that classifies traffic into application classes with another that classifies traffic into group classes. Moreover, it allows to perform a higher-level comparison between two classifiers that both use application classes, by looking at differences only in terms of groups.

To build a valid application database inside TIE, we started by analyzing those used by the CoralReef suite [77], and by the L7-filter project [80], because they represent the most complete sets that are publicly available and because such tools represent the state of the art in the field of traffic analysis and classification tools. By comparing such to application databases, we then decided to create a more complete one by including information from both sources and trying to preserve most of the definitions in there.

To each application class, as shown in Figure 4.9, TIE associates the following information:

- An integer application identifier that unically identifies the application.
- A human readable label to be used for readable output.
- A group identifier that associates the application to a category.

Moreover, to introduce a further level of granularity, inside each application class we allow the definition of sub-application identifiers in order to discriminate among sessions of the same application generating traffic with different properties(e.g. signaling vs. data, or Skype voice vs. Skype chat, etc.). To each sub-application the following information is associated:

- A sub-application ID.
- A human readable label to be used for readable output.
- A long description.

Each application class has at least the default generic sub-application ID "0".

To obtain an easily manageable and portable application database we adopted an ASCII file format. Figure 4.10 shows portions of the *tie_apps.txt* file. Each line defines one application identified by the pair $(AppID, SubID)$.

To properly define the application groups we started from the categories proposed by [72] and then we extended them by looking at those proposed by CoralReef [77] and L7-filter [80]. The resulting database, as shown in figure 4.11, uses the same format adopted for the applications database file and contains a label and a description for each group. The association of

```
typedef struct sub_app {
    char *sub_label;
    char *descr;
} sub_app;

typedef struct app {
    char *label;
    u_int8_t group_id;
    sub_app *sub_id;
    u_int8_t sub_id_count;
} app;
```

Figure 4.9: TIE: structure for application classes.

each application to a specific group has been done following the definitions given in [72] and [77].

The main output file generated by TIE contains information about the sessions processed and their classification. The output format is unique but semantics depend on (i) the operating mode in which TIE was run and (ii) the session type. The output file, by default called *class.out*, is composed by a header and a body. The header contains details about the whole traffic results, the plugins activated, and the options chosen. The body is a column-separated table whose fields are described in the following:

```
#AppID SubID GroupID Label      SubLabel         Description
0,     0,    0,      "UNKNOWN", "UNKNOWN",       "Unknown application"
#
1,     0,    1,      "HTTP",    "HTTP",          "World Wide Web"
1,     1,    1,      "HTTP",    "DAP",           "HTTP by Download Accelerator Plus"
1,     2,    1,      "HTTP",    "FRESHDOWNLOAD", "HTTP by Fresh Download"
[...]
1,     7,    1,      "HTTP",    "QUICKTIME",     "Quicktime HTTP"
[...]
4,     0,    1,      "HTTPS",   "HTTPS",         "Secure Web"
5,     0,    9,      "DNS",     "DNS",           "Domain Name Service"
[...]
10,    0,    3,      "FTP",     "FTP",           "File Transfer Protocol"
10,    1,    3,      "FTP",     "FTP_DATA",      "File Transfer Protocol (data stream)"
10,    2,    3,      "FTP",     "FTP_CONTROL",   "File Transfer Protocol (control)"
[...]
```

Figure 4.10: TIE: definitions of application classes from the file *tie_apps.txt*.

```
#GID Label                Description
0,   "UNKNOWN",            "Unknown group"
1,   "WEB",                "World wide web"
2,   "MAIL",               "Mail"
3,   "BULK",               "File transfer"
4,   "MALICIOUS",          "Malicious applications (trojan,worm,virus, attack)"
5,   "CONFERENCING",       "Conferencing and chat"
6,   "DATABASE",           "Database"
7,   "MULTIMEDIA",         "Multimedia (audio/video streaming)"
8,   "VOIP",               "Voice over IP"
9,   "SERVICES",           "Generic services"
10,  "INTERACTIVE",        "Interactive (login/remote control)"
11,  "GAMES",              "Games"
12,  "P2P",                "Peer-to-peer"
13,  "GRID",               "Grid"
14,  "NETWORK_MANAGEMENT", "Network management"
15,  "NEWS",               "News"
16,  "FILE_SYSTEM",        "File system"
17,  "ENCRYPTION",         "Encryption"
18,  "TUNNELING",          "Tunneling"
```

Figure 4.11: TIE: file format for definitions of group classes.

- **id**: the session identifier
- **5-tuple**: transport layer protocol, source and destination addresses and ports
- **timestart**: timestamp of the start of the session
- **timeend***: timestamp of the end of the session
- **pkt-up***: number of upstream packets
- **pkt-dw***: number of downstream packets
- **bytes-up***: number of upstream bytes
- **bytes-dw***: number of downstream bytes
- **app_id**: application identifier resulted from classification
- **app_subid**: application sub-identifier resulted from classification
- **confidence**: confidence value of classification process

The output format is the same for all the operating modes, but the semantics of the fields marked with an asterisk changes. In *offline* mode those fields refer to the entire session. In *realtime* mode they refer only to the period between the start of the session and the time the classification of the session has been made. This is done in order to reduce computations to the minimum after a session has been classified. Finally, in *cyclic* mode an output file with a different name is generated for each time interval, and fields marked with an asterisk refer only to the current interval.

```
# tie output version: 1.0 (text format)
# generated by: ./tie -P 20 -t 125

# 2 plug-ins enabled: l7 port

# begin trace interval: 1221921072
# trace interval duration: 300 s

#id src_ip    dst_ip     proto sport dport dwpkts uppkts dwbytes upbytes t_start           t_last            app_id sub_id confidence
7   10.0.0.55 10.0.0.129 17    4672  4672  1      1      19      48      1221921072.799580 1221921072.892036 40     0      25
5   10.0.0.55 10.0.0.209 17    4672  4672  1      1      19      225     1221921072.799178 1221921073.033699 40     0      25
12  10.0.0.54 10.0.0.80  17    33332 53    1      1      124     34      1221921073.257437 1221921073.274311 5      0      50
47  10.0.0.55 10.0.0.151 17    4672  4672  1      1      19      48      1221921074.989144 1221921075.108251 40     0      25
51  10.0.0.55 10.0.0.57  17    4672  4672  1      1      169     35      1221921075.039750 1221921075.254110 0      0      0
40  10.0.0.55 10.0.0.125 6     2094  4662  1      1      92      108     1221921074.984972 1221921075.299248 127    0      50
248 10.0.0.54 10.0.0.67  6     38629 1863  1      1      8       5       1221921088.905082 1221921089.114479 57     0      50
```

Figure 4.12: TIE: an example of classification output file.

4.8 Port-based classification plugin

The Port-based classification plugin relies on source and destination port numbers as features, and works on both TCP and UDP protocols. Because several tools performing port based classification were available, we tried to reuse the most up-to-date. In our search the CoralReef suite [77], developed by CAIDA, was the one with the largest and up-to-date port-based application database. To "not reinvent the wheel" and to conform to state of the art, the classification plugin we implemented from scratch relies on the CoralReef signature file called *Application_ports_master.txt*. Figure 4.13 shows some parts of it. As shown, for each application some fields are defined, among the most important are:

- **name**: it univocally identifies the application
- **group**: it associates each application to a category
- **sport**: the transport layer source ports
- **dport**: the transport layer destination ports
- **protocol**: the transport layer protocols

The last three fields can contain a comma separated list of integer numbers and ranges (e.g. "$50-75$" corresponds to values from 50 to 75). Moreover the jolly character "*" matches any value.

To import such signatures in the Port-based classification plugin, we implemented a simple parser that retrieves only the above mentioned fields and stores them into a hash table, in which the generic element has the structure shown in figure 4.14. Being that TIE determines the direction of a session differently compared to CoralReef (i.e. we consider the source port the one from the host generating the first packet), our parser swaps source and destination ports. Moreover, because TIE manages applications using an integer identifier key, the parser does the mapping of each application by looking at

```
[...]

# ------------------------------------------------------------
description:    World Wide Web
name:   HTTP
group:  WWW
sport:  80,8080
dport:  *
sym:    1
protocol:       6
priority:       10
contributor:    bigj
date:   1999-07-08
reference:      IANA Port assignments
url:    http://www.iana.org/assignments/port-numbers

[...]

# ------------------------------------------------------------
description:    Post Office Protocol (v2 & v3)
name:   POP
group:  Mail/News
sport:  109-110,995
dport:  *
sym:    1
protocol:       6
priority:       10
contributor:    rkoga
date:   2001-03-16
reference:      IANA Port assignments
url:    http://www.iana.org/assignments/port-numbers

[...]
```

Figure 4.13: CoralReef file format for port-application definitions.

its label. At the moment there is not a dedicated file to manage these associations, because TIE application database was built starting from the one defined by CoralReef, thus having the same labels. In the future we plan to implement such file to avoid modifications to the TIE application database when *Application_ports_master.txt* changes.

When a signature contains port ranges or more source-destination combinations, the parser creates an entry in the hash table for each of them. This approach speeds up the classification process at the expense of few additional bytes of memory. Moreover to manage jolly characters inside port fields we use the 0 value, because it is not a valid port number.

```
/*
 * This is the structure used to store port info
 */
typedef struct port_info {
        u_int16_t sport;                        /* source port (key) */
        u_int16_t dport;                        /* destination port (key) */
        u_int8_t proto;                         /* protocol type (key) */
        u_int16_t app_id;                       /* application ID */
        u_int8_t app_subid;                     /* application sub ID */
} port_info;
```

Figure 4.14: TIE: element of the port information hash table.

The algorithm implemented by the classifier on each session is very simple. It performs three lookups into the hash table by specifying the following information combinations:

- transport protocol and both source and destination ports
- transport protocol and destination port only
- transport protocol and source port only

The lookup, if successful, will return the corresponding entry containing the application identifier. The confidence value is always set to 100 when a hit occurs or set to 0 otherwise.

In Chapter 5, Section 5.4, we use the Port-based classification plugin as a base reference when experimentally evaluating and comparing performance,

in terms of speed, cpu and memory utilization, of other two classification plugins. Results show that the Port-based classification plugin is the fastest and the least cpu-intensive technique with also the smallest memory footprint.

4.9 L7-filter classification plugin

Another classification plugin distributed with TIE is based on a deep payload inspection technique. We chose to implement the same technique used by L7-filter [80] inside a TIE classification plugin for the following reasons: (i) we wanted to support at least one payload-inspection technique to compare it against completely different approaches; (ii) among the publicly available tools, L7-filter is one of the most popular; (iii) we needed to implement at least one ground-truth technique in TIE, and L7-filter routines are often used in literature for ground-truth [99] [127]; (iv) the current version of L7-filter is not easy to use on traffic traces. Indeed, because of its nature, L7-filter natively works only on Linux platforms and can only analyze traffic from a network interface. The only way to run it on previously-captured traffic is to replay that traffic (e.g. using tcpreplay [128]) on a network interface. Unfortunately such trick does not allow to work at high traffic rates ($\sim$ 1 Mbps), thus practically limiting its application to small traffic traces. By supporting the same technique under TIE we do not have these limitations anymore and we can run it both under Linux and FreeBSD.

L7-filter is an open-source project for Linux and it is available in two different versions: kernel and user-space. The original project was born in kernel space, where many functionalities are implemented by the Netfilter [129] framework, the same used by *iptables* to provide firewalling, NAT (Network Address Translation) and packet mangling under Linux. The user-space version, currently in a early stage of development, gets data through Netfilter's queues and implements connection tracking from scratch.

The L7-filter classification technique uses regular expressions. The kernel version is limited only to simple regular expressions, whereas the user-space version uses the GNU ones. A regular expression (or regexp, or pattern) is a

rule, in the form of a text string, describing set of strings. In general a regexp r matches a string s if s is in the set of strings described by r. A regexp can contain printable characters and some operators among the following:

- **The Match-any-character Operator**: represented by the "." character, it specifies the presence of a generic character.
- **The Concatenation Operator**: obtained by juxtaposition, it concatenates printable characters.
- **Repetition Operators**: represented by "∗+?{}" symbols, they allow to specify how many times a character sequence/group has to appear.
- **The Alternation Operator**: represented by the "|" symbol, it allows to specify alternatives inside a single regexp.
- **List Operators**: represented by a list of characters appearing inside square brackets ("[...]"), they allow to specify a list of characters to match.
- **Grouping Operators**: represented by a list of characters and operators appearing inside brackets ("(...)"), they allow to aggregate them into groups.
- **The Back-reference Operator**: represented by a "\ < *digit* >" sequence, it allows to refer to a group appearing inside the same regexp.
- **Anchoring Operators**: represented by "^$" symbols, they allow to specify respectively the start and the end of the string.

For instance, the regex:

```
^ssh-[12]\.[0-9]
```

matches the first characters of a SSH connection, where the initial "*ssh-*" string is followed by the version number. As specified in the pattern it could be 1.x or 2.x, where x is a digit from 0 to 9.

L7-filter during its startup loads the application patterns from several text files with "pat" extension. A pattern file is composed by several lines as follow:

- empty and blank lines are skipped
- lines starting with "#" are comments
- the first non-comment line must be the name of the application
- the next non-comment line must be the actual regular expression
- optionally a different regular expression can be specified for the userspace version if the line begins with "userspace pattern=“
- optionally some flags can be specified to be used by GNU regexp if the line starts with "userspace flags=“

Figure 4.15 shows the file that defines the regexp associated to the Bittorrent protocol.

After loading signatures, L7-filter processes packets by collecting the payload of each session into an array, independently of its direction, and removing the null bytes. Removing null bytes is necessary to pattern matching, because the regular expression engine uses null-terminated strings. The matching process is triggered by the reception of a packet carrying payload and if no match is found the session is left unclassified. If after 10 packets the session can not be classified, then it will be set as *Unknown* and its subsequent packets are ignored.

To develop a TIE classification plugin implementing the same technique used by L7-filter (TIE-L7) we started from the latest code-repository checkout of the user-space version. It was necessary to adapt some aspects of the TIE platform to work with this plugin. First, we had to add to TIE an option to make several classification attempts for the same session, by modifying the Decision Combiner. In fact, L7-filter tries to classify a session more than once: only if the regexp match fails after the reception of

the first 10 packets, then the session is marked as *Unknown*. Moreover, to let TIE work with the same definition of session, it was necessary to implement some heuristics to follow the state of TCP connections. We analyzed the heuristics implemented in the user-space version of L7-filter, that simply assume that a packet carrying the FIN flag determines the expiration of a TCP session, and added them as optional. During this study we also identified few bugs in the user-space version of the code that we reported to the developers. Moreover, as explained in Section 4.4 we added more heuristics for TCP connections that can be optionally activated. The pattern matching routines did not need any change to be ported to TIE's classification plugin. However, in order to support the FreeBSD operating system we had to include the GNU pattern matching libraries into the plugin package, because the implementation of such libraries under several versions of this operating system is extremely slow. Furthermore, in order to integrate into TIE the pattern files containing signatures used by L7-filter, it was necessary to

```
# Bittorrent - P2P filesharing / publishing tool - http://www.bittorrent.com
# Pattern attributes: good slow notsofast undermatch
# Protocol groups: p2p open_source
# Wiki: http://www.protocolinfo.org/wiki/Bittorrent
#
# This pattern has been tested and is believed to work well.
# It will, however, not work on bittorrent streams that are encrypted, since
# it's impossible to match encrypted data (unless the encryption is extremely
# weak, like rot13 or something...).

bittorrent

# Does not attempt to match the HTTP download of the tracker
# 0x13 is the length of "bittorrent protocol"
# Second two bits match UDP wierdness
# Next bit matches something Azureus does
# Ditto on the next bit.Could also match on "user-agent: azureus", but that's in the next
# packet and perhaps this will match multiple clients.

# Recently the ^ was removed from before \x13.  I think this was an accident,
# so I have restored it.

# This is not a valid GNU basic regular expression (but that's ok).
^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=)|d1:ad2:id20:|\x08'7P\)[RP]

# This pattern is "fast", but won't catch as much
#^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=)
```

Figure 4.15: L7filter bittorrent pattern file

port the L7-filter parser into the plugin and to associate each application to the corresponding TIE identifiers. To set such association, at start-up, TIE-L7 loads the signatures reading the list from a configuration file, which also contains associations between each application name and the corresponding $(AppID, SubID)$ pair.

Finally, to state the equivalence of TIE-L7 with the original L7-filter (the user-space version) we added to both softwares few routines to output debug information. Such output reports the list of the sessions detected and the corresponding classification result. We performed tests on several traffic traces, and after fixing small differences, we verified that TIE-L7 and L7-filter produced the same output. In Chapter 5 we show several experimental studies related to TIE-L7 by using large traffic traces. Specifically, in Section 5.3 we analyze in detail what are the portions of sessions (in terms of both packets and bytes) in which the successful regexp matches happen, and we find that most of them are related to the first few bytes of the first packet exchanged by two hosts. In Section 5.4 we analyze and compare the performance, in terms of speed, cpu and memory utilization, of TIE-L7 against two different classification plugins, and we find that it is possible to perform a much faster and less computationally-intensive payload inspection trading classification accuracy. Finally, in Section 5.5 we compare TIE-L7 against another ground-truth tool, and we find that such tools fail in identifying the entire traffic of the considered traces and often contradict each other.

4.10 TIE and the Research Community

TIE is a community-oriented tool, that is, it has been designed to allow the scientific community to easily develop real implementations of classification techniques to be evaluated by anyone on real (and live) traffic and fairly compared. However, besides community needs and deficient aspects of the state of art, during the design of TIE and its development, we have constantly payed attention to what the scientific community had already produced, both in terms of functionalities and data definitions/formats. Few examples follow:

- TIE uses the Libpcap library for live traffic capture and trace management, which is a *de facto* standard supported by most common operating systems. The vast majority of the traces made publicly available by the scientific community are in Libpcap (tcpdump) format, this makes them immediately usable by TIE.

- TIE supports different definitions of *sessions* according to those produced in literature (see Table 3.1 in Chapter 3).

- In the definition of classes and class IDs, we have carefully considered definitions already used by the most popular tools (e.g. CoralReef from CAIDA [77] and the Linux project L7-filter [80]. Moreover we have created a class hierarchy made of application groups, applications, and application sub-IDs, in order to represent the different types of classes considered in literature and to allow comparison even when they differ (e.g. approaches classifying applications against approaches classifying categories of applications).

- In the implementation of the first classification plugins we adopted definitions and algorithms widely used and accepted, as the CoralReef file of rules for port-application associations in the case of the Port-based classifier, and L7-filter algorithm and signatures in the case of the TIE-L7 classifier.

Furthermore, because designed as a community-oriented tool, TIE was involved since prototype stage into collaborative projects with other research groups. RECIPE ("Robust and Efficient traffic Classification in IP nEtworks") is a national project of two years (2007/2008) funded by the Italian Ministry of University and Scientific Research (PRIN 2006 Research Programs) [131]. Five research units grouping researchers from seven Italian universities participate to this project, which is is about the development of efficient techniques and tools for network traffic classification. TIE was presented in several RECIPE meetings and distributed to project partners, and it has become the reference tool in the project as regards development

Table 4.1: TIE classification plugins available and under development. The table highlights input from the community and joint activities.

Classification Plugin	Features based on	Classification approach	Status	Collaborations and contributions from the community
Port	Protocol ports	Port-based	Available	Developed by UNINA, signatures from CAIDA [77]
L7	Payload	Deep payload inspection	Available	Developed by UNINA, code and signatures from Linux L7-filter [80]
NBC	Payload	Lightweight Payload Inspection (see Section 5.4)	To be released	Developed by UNINA
GMM-PS	First few packet sizes	Gaussian Mixture Models [97]	Under test	Developed by UNINA
HMM	Packet size and inter-packet time	Hidden Markov Models [130]	Under devel.	Development by UNINA
FPT	Packet size and inter-packet time	Statistical [99]	Under devel.	Joint work between UNINA and University of Brescia in the context of the RECIPE research project [131]
Joint	Packet size and inter-packet time	Nearest Neighbour	Under devel.	Joint work: UNINA, CAIDA, Seoul National University
GT	Information from Hosts	Ground-Truth	In early devel.	Joint work: University of Brescia, CAIDA, UNINA

of traffic classification implementations and their experimental comparison. The development of a classification plugin implementing the technique presented in [99] for example, is part of a joint collaboration between the research group on computer networks at University of Napoli Federico II and the telecommunications group at University of Brescia (see Table 4.1).

NETQOS is a European Specific Targeted Research Project (STREP) from the 5th call of IST FP6 framework, contributing to the strategic objective of "Research Networking Testbeds" [132]. The NETQOS project is about the development of an autonomous policy-based QoS management approach for heterogeneous communications networks, in order to provide enhanced end-to-end QoS and efficient resource utilization. By addressing automation of network level policy management, the project allows for dynamic adaptation of the managed systems in response to changes of requirements in the operational environment. Users and applications are allowed to dynamically change their Quality of service (QoS) requirements while maintaining a smooth delivery of the required QoS. In the NETQOS framework,

two possible categories of applications are considered: NETQOS-aware applications, which communicate with several components of the architecture when they are launched in order to active QoS negotiation and management and measurement tasks, and NETQOS-unaware applications. The latter are applications that do not directly interact with NETQOS components. In order to provide adequate QoS to the user, and in general to enforce the appropriate network/transport level policies, NETQOS-unaware applications must be identified by looking at their traffic. TIE has been integrated into the NETQOS framework as a component for online classification of traffic generated by NETQOS-unaware applications. We used TIE configured in *realtime* mode and added functionalities to notify classification results to other NETQOS components in a specified format. The integration of TIE into a complex architecture like NETQOS was successful and was validated also during a formal demo session of the project.

Moreover, TIE has been recognized as reference tool in the European COST Action IC0703 "Data Traffic Monitoring and Analysis (TMA): theory, techniques, tools and applications for the future networks" (shortly COST-TMA) [133]. Started in early 2008, COST-TMA aims at coordinating participants (more than 40, divided in network operators and research groups active in the field of traffic monitoring and analysis) to promote the development of novel techniques and to focus research efforts towards commonly recognized problems, thus driving research towards real-world applications. One of the three working groups of the project is devoted to traffic characterization and identification. TIE was recently presented at the second COST-TMA meeting, held in September 2008, where it earned significant interest and was elected reference tool for future activities inside COST-TMA regarding traffic classification. At the time of writing discussions about joint activities with other research groups have already started.

Table 4.1 summarizes TIE classification plugins that are both available or under development and highlights connections with the research community. Part of the contributions presented in the next chapter involve the use of

some of the plugins reported in the table.

Chapter 5

Contributions to Traffic Classification

5.1 Introduction

In Chapter 3 we presented a critical analysis of the literature and the state of art in traffic classification, highlighting several important open issues in this field. The development of TIE, expounded in Chapter 4, was done in order to fill one of the large gaps present in this research field, by offering an instrument to the scientific community for testing and evaluating practical implementations. However, there are other issues in traffic classification exposed in Chapter 3 that we address in this chapter by investigating new classification techniques and analyzing limitations of the current ones, both through experimental analyses. Moreover, we point out that part of the experimental work presented in this chapter has been carried out thanks to TIE and some classification plugins written on purpose.

A careful analysis of literature indicates that there is still need to investigate new techniques alternative to payload inspection. The first contribution of this chapter is a classification technique based on statistical features that are totally novel and are based on the understanding of packet-level properties of traffic through several studies (that are discussed in Chapter 2). Moreover, in the experimental analysis of such classification technique, we follow the recommendations expressed in Chapter 3: we conduct an experimental

analysis of our technique considering overall link traffic, testing our approach on recent traces, and reporting results in terms of both byte-accuracy and of metrics widely adopted in the field of machine-learning and pattern recognition. In Chapter 3 we observed that the only implementations of traffic classifiers currently available are based on payload inspection. Such approaches suffer from limitations that discourage their use or even make it impracticable, such as the need to access to full packet content and the computational load. We therefore study these limitations of payload inspection approaches and we come up with a lightweight modification to deep payload inspection which goes towards the directions of *online* classification and *multi-classifier* systems (see Chapter 3 Section 3.6). We evaluate the proposed approach by implementing it as a TIE classification plugin and comparing it against a largely used deep-payload inspection classifier and a port-based classifier, both of them also implemented as TIE plugins. We conclude the chapter by investigating, for the first time in literature, the problem of accuracy of *ground-truth* systems (also highlighted in Chapter 3 Section 3.6). Indeed, despite the fact that such systems are used as the reference when evaluating the classification techniques that are being proposed, there is not much knowledge about them and their reliability. We carry out an experimental evaluation by comparing two largely used *ground truth* systems based on payload inspection. Results show how both are not able to identify the entire traffic present on a link and how sometimes they contradict themselves. Such results on one side demonstrate inaccuracy of payload-based classifiers, on the other urge the scientific community to accurately verify and improve the systems used for reference when studying traffic classification.

5.2 Traffic Classification through Joint Distributions of Packet-level statistics

The experimental analyses reported in Chapter 2 show distinctive properties of network traffic from different applications when traffic is analyzed at packet-level, that is, in terms of Packet Size (PS) and Inter-Packet Time (IPT). The classification technique presented in this section is heavily based on such studies. Indeed, as will be explained in detail in the following, to produce features for traffic classification we considered the joint distribution of PS and IPT and we applied a strong discretization to the estimate of their joint PDF. As regards the classification technique used to process such features, we considered machine-learning algorithms like the K-Nearest Neighbor and SVM. The major contribution of this study is represented by the introduction of effective packet-level features, which, as far as we know, have never been previously proposed in literature. We show that, in conjunction with a machine-learning classification algorithm, such features allow to build a traffic classifier that looks promising in terms of robustness to evasion from identification[1].

5.2.1 Traffic view and statistical features

We decompose network traffic into *biflows*, which represent an extension of flows by considering traffic in both directions, so that the $\{src_{ip}, src_{port}\}$ and $\{dst_{ip}, dst_{port}\}$ pairs can be swapped (see Chapter 3 Section 3.4). The timeout (of 90 seconds) is evaluated by considering packets in both directions. We call the two directions *upstream* and *downstream*, where the first one refers to packets sent by the host sending the first packet. Packets sent by the other host belong to the downstream direction. Biflows can be seen as a very simple heuristic for TCP connections and an approach to aggregate packets into udp "sessions". With the purpose to obtain better classification

[1]This research activity has been carried out in the context of a collaboration with the Cooperative Association for Internet Data Analysis (CAIDA), University of California San Diego, CA, USA, and Seoul National University, Korea.

accuracy, we classify biflows instead of flows. This choice allows to exploit the obvious correlation between traffic in both directions, since generated by the same network application. Even if it is not always possible collecting traffic in both directions (e.g. fiber links on a backbone), it is straightforward to apply the same approach to unidirectional traffic information, but with a probable degradation of classification accuracy.

In Chapter 2 we observed that network applications exhibit distinctive behaviors for marginal distributions and autocorrelations of PS and IPT. Specifically, we found a strong invariance of average profiles of PDFs for each different application when looking at traces coming from different links and taken at different times (*space and time invariance*) [50] [134] [48]. By "average profiles" we mean that the PDFs of thousands flows have been averaged to a single PDF, however we observed this behavior also when looking at single profiles (i.e. separately considering the PDF of each single flow). Moreover, we found that IPT and PS of the same packet are usually very correlated. This can be taken into account by considering the Joint distribution of PS and IPT. We therefore developed a set of features based on this observation to build traffic fingerprints of different network applications that could be processed by a machine-learning algorithm. Because we consider biflows as objects to be classified, a *fingerprint* of the application will be built by considering the joint distributions of both directions[2].

The machine-learning algorithms targeted however need as input-features a discrete set of data. We therefore need to identify a binning criterion for the joint distributions of IPT and PS. An ideal grid is overimposed to the plane identified by the PS axis and the IPT axis. A bin corresponds to each cell of the grid. The normalized "height" of each bin is a feature.

Basing on our studies in traffic analysis (see Chapter 2) we applied a non-uniform binning both to reduce the number of features to be considered

[2]Since, as shown later in this section, this approach looks successful, we are considering as a future work to further develop the set of features used by adding to the joint distributions of PS and IPT of each direction, information regarding behavior on the opposite direction (e.g. number of bytes transmitted in the opposite direction before the considered packet was received).

(e.g. to reduce the computational complexity) and because we identified that distinctive properties can be grouped into ranges of values made of different sizes. For example, for the packet size parameter we considered the following upper boundaries (in bytes) for the binning: {2,5,10,100,200,500,1000,1400,∞}. Whereas for IPT we considered the following upper boundaries (in microseconds): {10,100,1000,10000,500000,1000000,10000000,∞}. Our traffic processing platform [62] assigns each observed packet to a biflow and to one of its two directions. It then updates the discretized joint PDF assigned to that direction by incrementing the counter associated to a specific bin. The correct bin is identified by evaluating the packet's PS and its IPT with respect to the previous packet from the same direction, and comparing such values against the aforementioned boundaries. At the end of the flow (or at program termination) the values inside the matrix of bin heights are normalized and dumped to a log file reporting other biflow information (unique ID, biflow-tuple, flow-level statistics, etc.).

We can visually represent the two upstream and downstream matrices assigned to a biflow as shown in Figure 5.1, where we consider the binned joint PDFs of a sample biflow. The height of each bin is indicated by the darkness of the corresponding cell and it represents the relative frequency of the packets into the associated range of PS-IPT values. The PS and IPT values, respectively on the x and y axes, refer to the upper boundaries of each cell.

In our experiments we also considered other biflow-level parameters as possible additional features like, for example, the number of packets transmitted for each direction and the biflow duration. Please refer to Section 5.2.4 for details. The matrices representing PS-IPT Joint PDFs however are the main features used in this work. To give an intuitive idea of how such features could really be used as *fingerprints* for classifying traffic, in Figures 5.2 and 5.3 we show random samples of Joint PDF respectively for the Edonkey application and for the Web application. From these figures we can note that samples from the two applications look quite different. Samples related

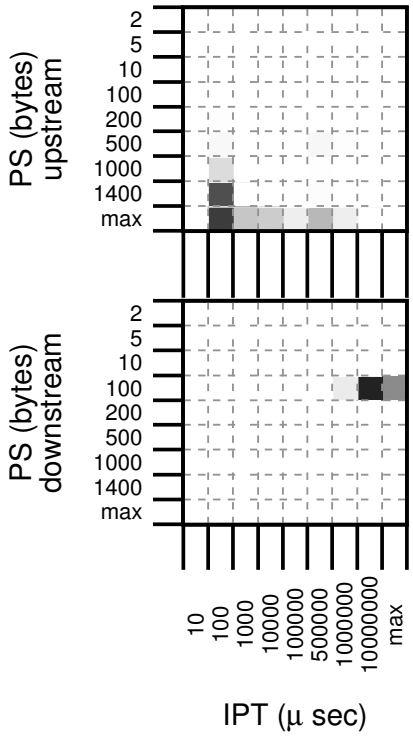


Figure 5.1: Sample matrices representing two Joint PDFs of PS-IPT

to Edonkey present an upstream traffic mainly constituted of full-size packets with a majority of small IPT. This traffic is probably associated to a data transfer. On the opposite direction, instead, we observe small-size packets with large IPTs of the order of tens of seconds. This can be referred to a sort of control channel (e.g. subsequent requests of chunks of files). Looking at Figure 5.3 we observe that samples (a) and (b) have a different behavior compared to (c) and (d). The first ones have an empty upstream distribution because the upstream direction was made of a single packet (thus IPT was not computable). In this case we are probably observing a single HTTP request (upstream) followed by the transfer of the requested file (downstream). Samples (c) and (d) instead are probably related to biflows with HTTP using *persistent connections*, in which subsequent requests are made inside the

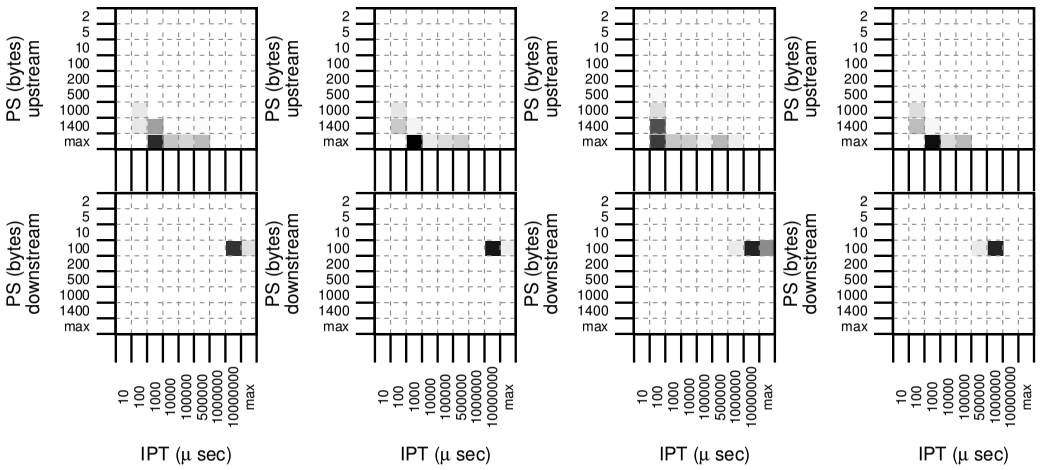


Figure 5.2: Sample fingerprints through joint PDFs of Edonkey traffic.

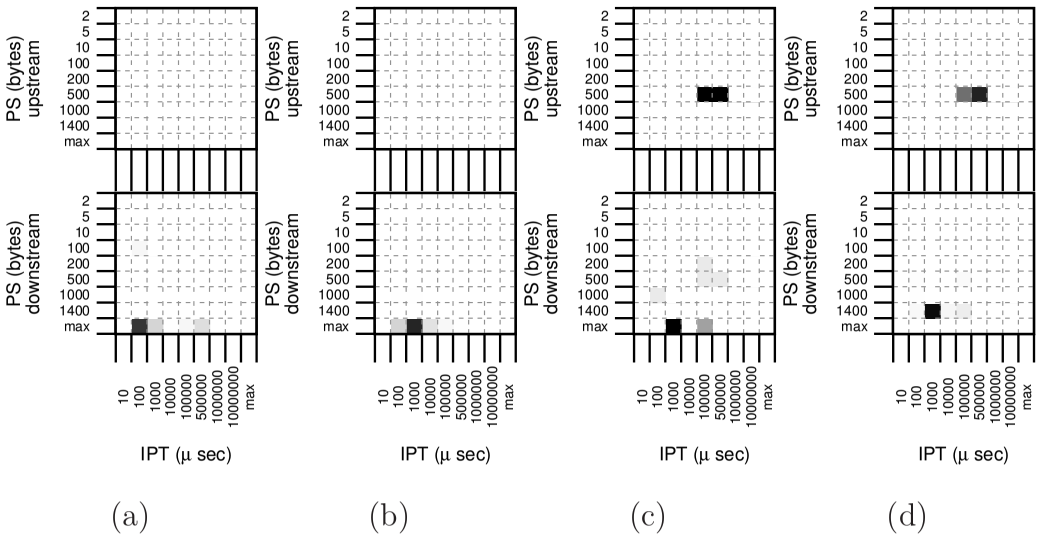


Figure 5.3: Sample fingerprints through joint PDFs of Web traffic.

same TCP connection. Medium-size packets (HTTP requests often do not fill an entire packet) and IPT of the order of fractions of seconds to few seconds (requests generated directly by the browser or caused by user clicks) are consistent with our hypothesis.

Features related to PS and IPT, and in particular related to their marginal distributions, for traffic classification purposes have already been presented in literature. However, to our knowledge, the approximation of the PS-IPT joint distribution represents a novel set of features that has never been evaluated. In [91], 249 possible discriminators for classification of traffic flows are listed. Among them we find features like mean and standard deviation of PS or of IPT, minumum, maximum, quartiles of their distributions. Some of these features have been used in several papers [70] [96] [114]. However, not only the detailed approximations of the distributions are never listed and considered, but such PS and IPT were never taken into account *jointly*. Summary statistics like mean, median and standard deviation obviously do not contain the same information (and thus discriminative power) of approximations of the PDF. For example, by averaging two possible modes that can be present into an applications: very small packets alternated to full-size packets are averaged into a mean PS of medium-size packets we loose a lot of information (even if variance can give us some hints). Moreover the joint characterization of PS and IPT carries even much more information than the two separate distributions. For example, without a joint modeling we could not distinguish between an application generating full-size packets with small IPT and small-size packets with large IPT, and an application that associates IPT and PS in the opposite way. The results shown in the next sections confirm indeed the discriminative power of the features here considered. Finally, as regards the use in literature of similar features for traffic classification, we clarify that the features used in [99] are totally different. The joint distributions considered in that paper, indeed, represent another kind of information: the authors consider the order of packet arrivals for all the biflows analyzed, and build a joint distribution for each category of packets depending on their

order of arrival, e.g. considering 5 packets per biflow would bring 5 joint distributions. The first of these joint distributions would therefore represent statistical properties of the *typical* first packet seen in the biflows of the application fingerprinted.

5.2.2 Machine-Learning Approach

To test our approach based on the use of the traffic features described above, we focus on machine-learning algorithms performing supervised learning. We suppose indeed that the traffic classes are defined and that we have availability of pre-classified data to train our classifier.

The objective of this study is to investigate the effectiveness of using packet-level features extracted from PS-IPT joint distributions and to identify an appropriate classification algorithm. Therefore, for our experimental analysis, we used the WEKA machine-learning software suite [101], often used in traffic classification studies [135, 93, 95, 136, 70, 115] because it supports a large set of highly configurable machine-learning algorithms. This way, we have large flexibility in experimenting with datasets and different algorithms. Only after the approach is well-defined and an appropriate machine-learning algorithm has been selected we can develop a prototype implementation as a TIE classification plugin. To separate training and testing sets, 50% of each considered trace is chosen randomly to form a pool of training flows, and the remaining 50% is used for a pool of testing ones. After experimenting with several machine-learning algorithms we restricted our final experiments only on K-Nearest Neighbor and Support Vector Machines because we achieved best results with them. Here we briefly describe such algorithms:

***k*-Nearest Neighbors (*k*-NN)** [69] computes Euclidean distances from each test instance to the k nearest neighbors in the n-dimensional feature space. The classifier assigns the majority class label among the k nearest neighbors to the test tuple. We use $k = 1$, 3, and 5. Where k determines the number of nearest training instances against which the algorithm checks the

Table 5.1: Characteristics of analyzed traces

Set	Date	Day	Start	Duration	Link type	Packets	Bytes	Biflows
KAIST	2006-09-14	Thu	16:37	21h 16m	edge	357 M	259 G	221K
UNINA	2008-05-16	Fri	10:42	18m	edge	52 M	40 G	42 K

distance with the sample under classification.

Support Vector Machines (SVM) [137, 138, 127] The principle at the base of SVM is to construct a separating hyperplane that maximizes the distance between two sets of vectors (each set pertaining to a class) in an n-dimensional feature space [138]. Pairwise classification can be easily extended to multi-class problems in several ways. The parameters of the separating hyperplane with the maxium distance are derived by solving an optimization problem. In our setup we used the Sequential Minimal Optimization (SMO) [139], which decomposes the optimization problem into several 2-dimensional sub-problems that can be solved analytically instead of requiring numerical optimization. Two important parameters in SVM are the complexity parameter C and the polynomial exponent p [127, 137]. We use 1 for both of them as in [137].

5.2.3 Traces and Datasets

Our datasets consisted of anonymized payload traces collected at two edge links located in Korea and Italy (Table 5.1). The KAIST trace was captured at one of four external links connecting a 1 Gb/s KAIST campus network and a national research network in Korea. The UNINA trace was captured at a 1 Gb/s link connecting one of UNINA campus networks to a national research network in Italy.

For establishing the *ground truth* in order to evaluate our classification approach we used Crl_pay, a classifier based on payload inspection but also adopting some heuristics, which has been developed on top of the CoralReef suite [77] and made available by CAIDA. Crl_pay has been originally used in [106] and [55] (besides more recent works) and details about the techniques adopted are given in [140]. Thanks to CAIDA researchers and interns Crl_pay

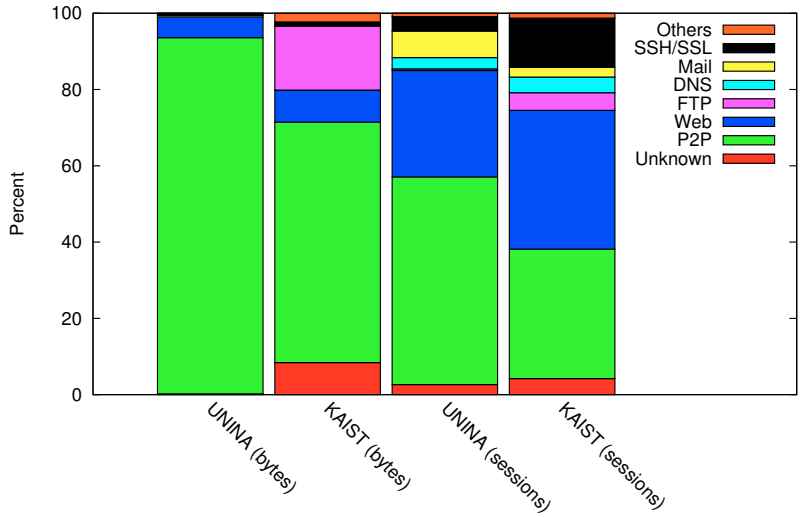


Figure 5.4: Application breakdown by percentage of biflows and bytes.

was recently augmented with more payload signatures from [87, 95, 141]. Moreover we verified several results from the classifier through manual payload inspection. The resulting ground-truth system includes payload signatures of various popular applications, summarized in Table 5.2. Since Crl_pay classifies *flows*, in order to obtain a list of pre-classified *biflows* we developed a simple procedure to merge its classification results to obtain biflows. The two class labels obtained for both directions matched in almost all cases, otherwise the conflicts have been solved by manual inspection.

We performed a filtering of data before training and testing machine-learning algorithms with them. We removed all biflows of very small size, most of them made of a single packet (for which therefore a joint PDF could not be built because of lack of IPT). Specifically, by removing all biflows with less than 10 packets for both directions we removed a high fraction of biflows (around 80%) from the data set that would only confuse (or make their work harder) the machine-learning algorithms, while still keeping 99.7% of total traffic in terms of bytes transferred. This was done with the purpose of (i) removing biflows generated by single packets, errors, scans, etc. (ii) reliev-

ing the load of the machine-learning algorithms (iii) focusing on the biflows that really weigh upon links traffic. Moreover, it is worth noting that more than 80% of the flows that were removed by the filtering were classified as unknown or uncertain by Crl_pay, confirming that such biflows could have been counterproductive for our tests. Such filtering had also the effect of almost totally removing the biflows classified as unknown by the ground-truth software and thus not usable for training and testing. However, we must note that by excluding such categories of traffic (e.g. scans), such approach as developed and tested here, cannot be considered for the identification of different kinds of attacks (e.g. port scans, scanning worms), since we are focusing on traffic generated by applications carrying data. Figure 5.4 shows payload classification results for our traces after applying the filtering. This application breakdown is shown by grouping the applications into the categories shown in Table 5.2, whereas the total number of distinct applications identified in each trace was around thirty.

Table 5.2: Application categories.

Category	Application/protocol
web	http, https
p2p	FastTrack, eDonkey, BitTorrent, Ares Gnutella, WinMX, OpenNap, MP2P SoulSeek, Direct Connect, GoBoogy Soribada, PeerEnabler
ftp	ftp
dns	dns
mail/news	smtp, pop, imap, identd, nntp
streaming	mms(wmp), real, quicktime, shoutcast vbrick streaming, logitech Video IM
network operation	netbios, smb, snmp, ntp, spamassassin GoToMyPc
encryption	ssh, ssl
games	Quake, HalfLife, Age of Empires, Battle field Vietnam
chat	AIM, IRC, MSN Messenger, Yahoo messenger
unknown	-

5.2.4 Experimental Results

We performed several sets of experiments, with different machine-learning algorithms and with different combinations of features. For the algorithms we considered: 1-NN, 3-NN, 5-NN, and SVM. The best results were always ob-

Table 5.3: Classification Performance Metrics for UNINA test set

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
P2P	0.944	0.072	0.943	0.944	0.944
Web	0.915	0.035	0.914	0.915	0.914
FTP	0.413	0.003	0.4	0.413	0.406
SSH	0.938	0.001	0.857	0.938	0.896
Mail	0.931	0.005	0.937	0.931	0.934
SSL	0.713	0.008	0.763	0.713	0.737
DNS	0.921	0.004	0.884	0.921	0.902
Chat	0.849	0.001	0.836	0.849	0.843

tained by the 1-NN algorithm, closely followed by 3-NN and 5-NN. Whereas the SVM algorithm achieved an overall classification accuracy few percentage points lower than the others. Therefore, unless specified, the results shown in this section refer to 1-NN with 50% of the data set used as training set. As regards the features used, we tried to add few features to the joint PDF. The use of transport-layer protocol ports, for example, increased overall accuracy of few points. This improvement is predictable, as protocol ports are still used by several legacy applications. However we removed ports from the feature sets because we wanted to test the effectiveness of the proposed approach when using only statistical informations about the biflows, and in particular we wanted to stress the ability of this novel typology of features to reach high values of classification accuracy.

The results shown in the following have been obtained with a feature set made of the values from the joint PDF matrices and with only two additional features: upstream-downstream packet ratio and biflow duration. For the first one, we used the ratio given by the number of upstream packets divided by the sum of upstream and downstream packets. As for the duration of the biflows, we applied a *log10* transformation to the values measured in milliseconds. These two features were added because such information related to the packets transferred cannot be derived by a joint distribution (e.g. the information on the number of packets is lost when computing the PDF). It is worth to note that overall accuracy does not decrease more than 2% when excluding these two features.

To measure the performance of classification on the test set, we use four metrics: *overall accuracy*, *precision*, *recall*, and *F-Measure*.

Table 5.4: Classification Performance Metrics for KAIST test set

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
P2P	0.913	0.05	0.909	0.913	0.911
News	0.931	0.001	0.901	0.931	0.916
Web	0.929	0.04	0.933	0.929	0.931
FTP	0.927	0.005	0.911	0.927	0.919
SSH	0.959	0	0.949	0.959	0.954
Mail	0.932	0.001	0.947	0.932	0.939
SSL	0.985	0.002	0.985	0.985	0.985
DNS	0.92	0.004	0.92	0.92	0.92
Streaming	0.676	0	0.742	0.676	0.708

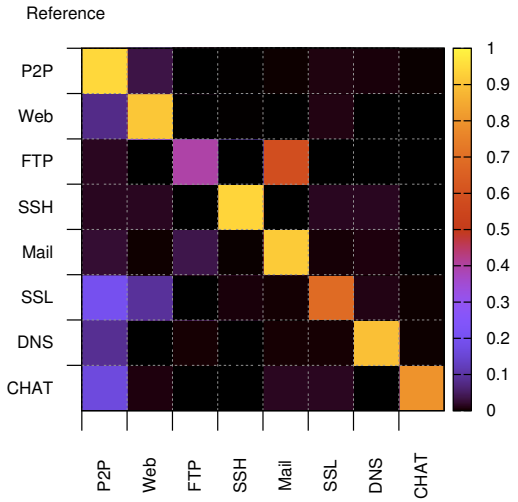


Figure 5.5: UNINA Trace: Confusion Matrix by group.

• *Overall accuracy* is the ratio of the sum of all True Positives to the sum of all the True Positives and False Positives for all classes.[3] We apply this metric to measure the accuracy of a classifier on the whole test set. The latter three metrics are used to evaluate the quality of classification results for each application class instead.

• *Precision* of an algorithm is the ratio of True Positives over the sum of True Positives and False Positives or the percentage of flows that are properly attributed to a given application by this algorithm.

[3]True Positives is the number of correctly classified biflows, False Positives is the number of flows falsely ascribed to a given application, and False Negatives is the number of flows from a given application that are falsely labeled as another application.

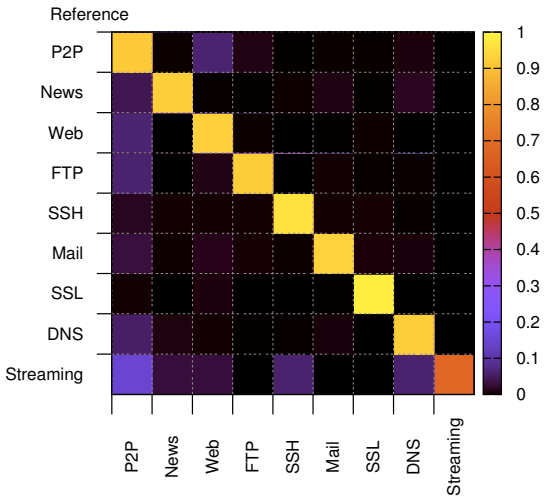


Figure 5.6: KAIST Trace: Confusion Matrix by group.

• *Recall* is the ratio of True Positives over the sum of True Positives and False Negatives or the percentage of flows in an application class that are correctly identified.

• Finally, *F-Measure*, a widely-used metric in information retrieval and classification [142], considers both precision and recall in a single metric by taking their harmonic mean: $2 \times precision \times recall/(precision + recall)$. This metric is useful to compare and rank the per-application performance of the different machine-learning algorithms considered.

Moreover, we compute overall accuracy also in terms of bytes (*byte-accuracy*). Because we are interested into evaluating the ability of the classifier to accurately identify the vast majority of traffic running on a link. Indeed, because biflows can be made of a very variable number of packets, it is important to evaluate accuracy by also considering the *weight* of each biflow (per bytes or per packets). We define overall byte-accuracy as the ratio of the sum of all bytes carried by True Positives to the sum of all bytes carried by the True Positives and False Positives for all classes. We do not report overall accuracy in terms of packets because values are always very

Table 5.5: Classification Performance Metrics for KAIST test set

Trace	Overall Accuracy	Overall Byte-Accuracy
UNINA	92.3%	98%
KAIST	92.9%	87%

close to those from byte-accuracy (only bytes from transport-level payload are included in our calculations). Finally, we also consider the *confusion matrix* to better understand classification results and to identify which kind of misclassifications most frequently happen.

Since the classification features here proposed, are closely related to the typical behavior of the applications in terms of the traffic that they generate, we expect that applications supporting the same kind of service behave similarly and thus present similar features. For this reason we grouped the applications considered into the categories reported in Table 5.2, each category corresponds to a class. As reported in Chapter 3, this approach has been adopted in several works presented in literature. The high values of overall accuracy shown in Table 5.5 confirm our intuition. Especially in the case of the byte-accuracy achieved for the UNINA trace, the tested approach is very successful in correctly classifying the entire traffic contained in our traces. We suggest that the difference in byte-accuracy between the two traces can be explained with the misclassification in the KAIST trace of few Peer-to-Peer biflows carrying large quantities of bytes. This hypothesis is consistent with the performance metrics of the *P2P* traffic class obtained for the two traces and reported in Tables 5.3 and 5.4.

In Figures 5.5 and 5.6 the confusion matrices for both data sets are represented in graphical form. Each row represents how a single class (reference on y axis) is classified by the algorithm (prediction on x axis). The confusion matrix is built by counting the biflows for each cell and by normalizing each row to 1. Values on the main diagonal represent the percentage of correctly classified biflows for each class. Both matrices show how the classifier performs excellently (yellow cells on the main diagonal) for almost all traffic categories. However, as for the KAIST trace, we note that the *Streaming* category is confused with *P2P* traffic in several cases. This is confirmed by

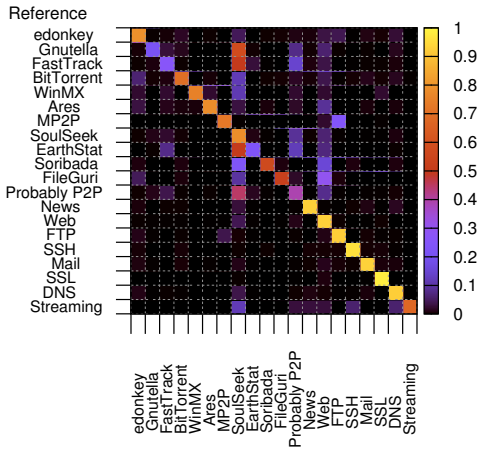


Figure 5.7: KAIST Trace: Confusion Matrix with P2P group exploded.

looking at Table 5.4, where *Streaming* is the only class with values of precision, recall, and F-Measure below 0.9. The worst performance, in the case of the UNINA trace, is achieved for the FTP category, which presents all three metrics around 0.4. Chat and SSL also show some problems. As regards FTP, the confusion matrix reveals that it is often misclassified as MAIL traffic. Our explanation of this phenomenon is that most of the misclassified biflows carry FTP signaling sessions, whose behavior is indeed very similar to some Mail sessions (e.g. POP) when they do not transfer much data. This hypothesis is confirmed by two observations: (i) the opposite misclassification does not happen: the *Mail* category is not often misclassified as *FTP*; (ii) despite of the considerably low value of precision for *FTP* traffic, the overall byte-accuracy reaches a very high percentage, which lets us think that the misclassified FTP biflows do not have a large byte-count.

We conclude the analysis of experimental results by showing, in Figures 5.7 and 5.8, the confusion matrices that we obtained when exploding the *P2P* traffic category and separately considering eleven different Peer-to-Peer file sharing applications plus the *Probably P2P* class. From such graphics we

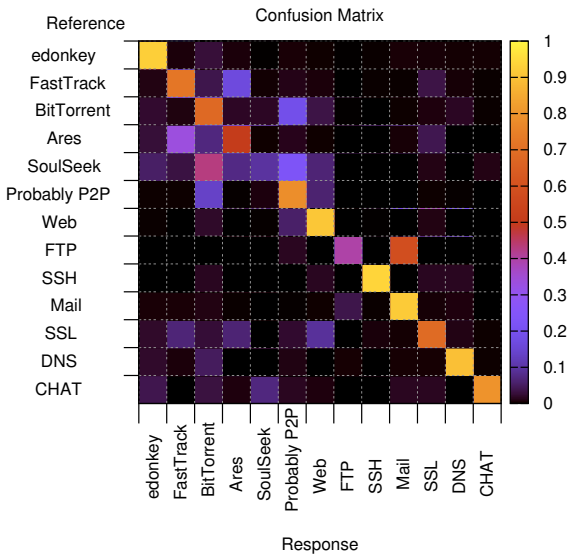


Figure 5.8: UNINA Trace: Confusion Matrix with P2P group exploded.

notice two things: firstly, the overall classification accuracy decreases when considering separate applications instead of grouping them into categories. Secondly, most of the added *confusion* when exploding the *P2P* category happens among classes associated to Peer-to-Peer applications. The fact that confusion happens within classes of that category (Peer-to-Peer) confirms that the classification approach here presented, and in particular the adopted traffic features, is able to catch the distinctive behavior of different categories of traffic rather than protocol or packet details that can be easily altered and are not necessarily shared by applications supporting the same kind of service (e.g. PS of the first 4 packets). We therefore look at such behavior as a symptom of robustness of the presented approach with respect to evasion techniques that change simple features (TCP flags, packet size of first few packets, string-based signatures) in order to confuse classifiers. We indeed plan to investigate this aspect in future works, also with the intent to understand the impact of traffic encryption on such classification features.

5.2.5 Discussion

In the evaluation of this novel traffic classification technique we tried to fulfill some of the recommendations that we pointed out in Chapter 3 (in particular in Section 3.6). We proposed an approach that: (i) does not use protocol ports or payload; (ii) it has been applied and evaluated by considering all the categories of traffic found on the observed links (e.g. not only TCP traffic); (iii) we used clear evaluation metrics commonly used by the machine-learning scientific community; (iv) we reported also the overall byte-accuracy metric (often neglected in literature); (v) we tested the approach on very recent traces (the UNINA trace is from mid 2008).

At the same time, however, such desirable properties make our classification technique difficult to be compared against the others presented in literature (in Chapter 3 we strongly advocated the thesis that current traffic classification literature lacks and needs comparisons). This is because most of the approaches proposed in literature present differences that do not make possible a fair comparison with ours: test on incomplete traffic, use of protocol ports, missing metrics, old traces, unreliable ground-truth, etc. We are currently working to implement the technique here presented as a TIE classification plugin, together with several known techniques in order to overcome this limitation (see Table 4.1 in Chapter 4. By now, looking at Table 3.1, the closest work to ours allowing at least comparison of metrics is [70], being the only one applied to the whole link traffic and reporting both session-accuracy and byte-accuracy. This work has already been described in Chapter 3, therefore here we just comment that it classifies flows, it is based on a Bayesian machine-learning approach, and among other features it uses also protocol ports (as mentioned, we do not use protocol ports as features, even if we verified that it increases classification accuracy). The best results there presented report an overall accuracy around 96% and byte-accuracy around 84%. Therefore, in terms of accuracy our approach is comparable to the one presented in [70], and even outperforms it when both are evaluated in terms of byte-accuracy.

Furthermore, we state that one of the most original contributions of this work is the use of features new in the context of traffic classification and which showed properties of robustness, that suggest their use in the design of approaches robust to traffic encryption and protocol obfuscation. Several classification techniques based on machine-learning, indeed, heavily rely on *weak* features, e.g. related to the very first packets [99] [70] [97]. These features can be easily altered with the purpose of obfuscation, whereas the features here proposed are more robust in this sense, as they are very dependent on the behavior of the application in terms of its traffic which is linked to the kind of service that it is supporting.

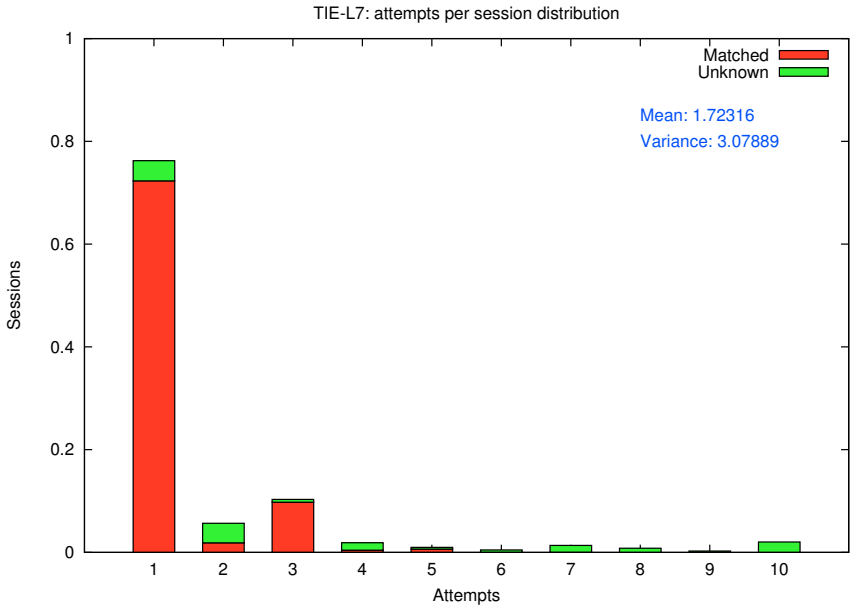


Figure 5.9: TIE-L7: Distribution of the number of attempts per session.

5.3 On the *Deepness* of Payload Inspection

Deep payload inspection approaches are used to classify network traffic by executing pattern-matching on the whole content of packets. In this section we perform an analysis aiming at understanding how *deep* into each single packet, and into each flow analyzed, a typical payload inspection classifier needs to go to identify applications associated to the traffic under observation. This is of interest to us because it represents a fundamental property, affecting (i) performance, (ii) privacy issues in traffic analysis and classification, (iii) methodologies for capturing traffic traces useful for studying traffic classification. We consider as reference L7-filter, because it is the state of art of available payload inspection techniques and it is used in many scientific works for ground truth creation [99] [127].

We have been able to perform this investigation by using the TIE classification plugin implementing L7-filter techniques (TIE-L7 in the following) introduced in Chapter 4. We added several debugging hooks to TIE-L7 to allow the collection of information related to:

- the number of attempts made for each session (TIE-L7 tries to match its regular expression rules against the stream of payload from a session every time a new packet is seen, by default up to the tenth packet);
- exactly "where" the matches of regular expression rules happened, in terms of (i) position of the packets inside the session, (ii) position of the matching string inside the payload stream and (iii) inside each packet.

In the following we show experimental results obtained by processing the UNINA trace described in Section 5.2. About 52 millions of packets were processed, TIE-L7 was invoked on a total of 394341 sessions and returned a match for 380244 of them. Figure 5.9 shows the distribution of the number of attempts per session. The red portion of the bars represents the percentage of sessions classified (the payload stream of the session matched against a regular expression rule). The green color indicates the unclassified sessions (no match happened); please note that they can result unclassified before the tenth attempt because such sessions end earlier. We highlight a first relevant result: the vast majority of the matches (72% of total attemps) happen at the first packet. The average number of attempts is indeed 1.7 with a variance of about 3. In general we observe that most of the matches happen before the fourth packet.

Figures 5.10 and 5.11 respectively represent the distribution of the *start* and *end* matching packets. A regular expression rule from TIE-L7 can indeed match a string in the payload stream whose parts were carried by separate packets. Such diagrams confirm that most of the matches start and end with the first packet, with observable percentages of start and end packets also at position two and three.

Considering a string that matches a regular expression rule spanning several packets, we can compute the offset of its first character and last character from the beginning of the packets respectively containing them. The distributions of these two offsets, adopting a bin size of 32 bytes, are depicted in Figures 5.12 and 5.13 respectively. They let us understand which portions of packets payload are really involved in successful matches of regular

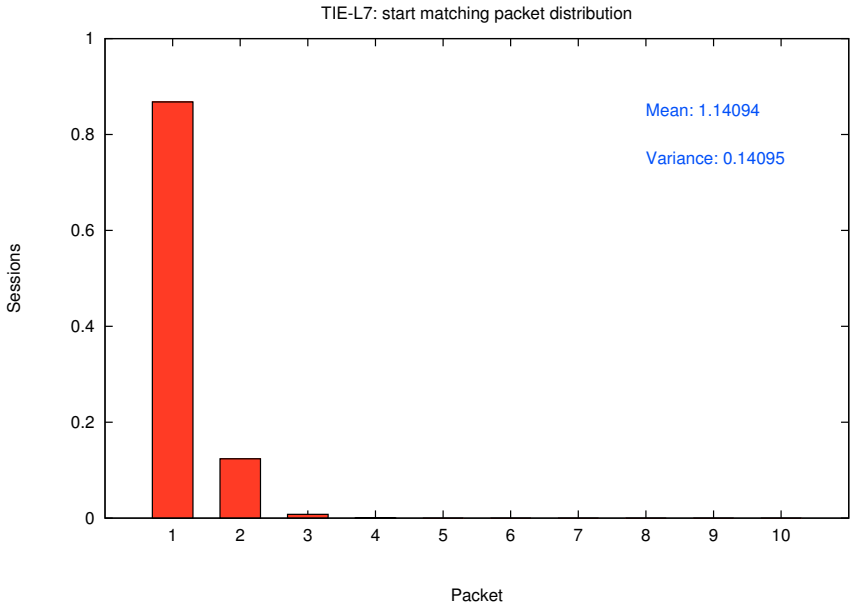


Figure 5.10: TIE-L7: Distribution of packet position when carrying the start of a matching string

expressions. Results are very interesting, in that they show how almost all matching strings start (99.98%) and finish (90.77%) in the first 32 bytes of payload. Specifically, the average offset inside its packet of the first byte matching a regular expression from TIE-L7 signature set is 0.06. Showing that the vast majority of matching strings start at the first byte of payload.

The findings reported in this section are interesting from several points of view. First, they furnish useful information for the tuning and the improvement of TIE-L7 / L7-filter and of deep payload inspection approaches in general. For example, L7-filter has an option to set the maximum number of packets to check (attempts to perform) before giving up on classifying a session. This is set by default to ten. From what we have found, depending on the specific requirements (processing time, access to traffic data, etc.) this parameter could be set to a lower value without significantly affecting the performance of classification accuracy. Moreover, from our experimental analysis we observed that there are rules in the signature set that always match packets in certain positions, or never match packets after a certain

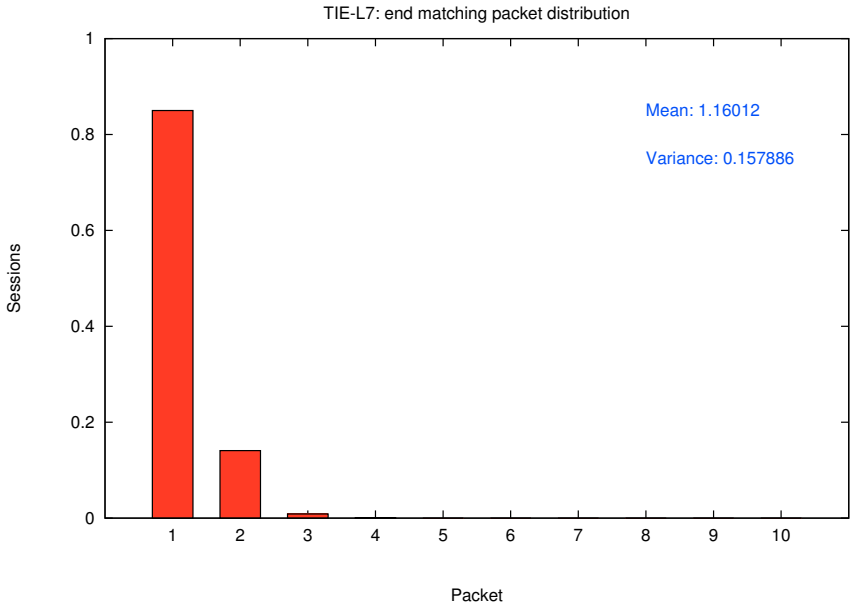


Figure 5.11: TIE-L7: Distribution of packet position when carrying the end of a matching string

position. Being that each new classification attempt is particularly onerous in pattern matching systems, a possible improvement to speed up the processing speed and to alleviate resource usage would be to add an optional "packet position" field to the signatures, in order to save unuseful attempts. An attempt destined to fail is indeed the most onerous, because all the the payload is going to be tested against all the available rules.

The analysis carried out here could be useful also as a guideline when setting up traffic capturing and archiving of traffic traces that involve the use of deep payload inspection. Often researchers are challenged by the decision on the quantity of payload of packets to preserve when capturing traffic. This usally involves a trade-off between accuracy and completeness of the data for experiments on a side and space constraints, logistics, and privacy issues on the other side.

Finally, we anticipate that the findings reported here have inspired the design of an alternative classification technique based on payload inspection, which will be illustrated in the next section.

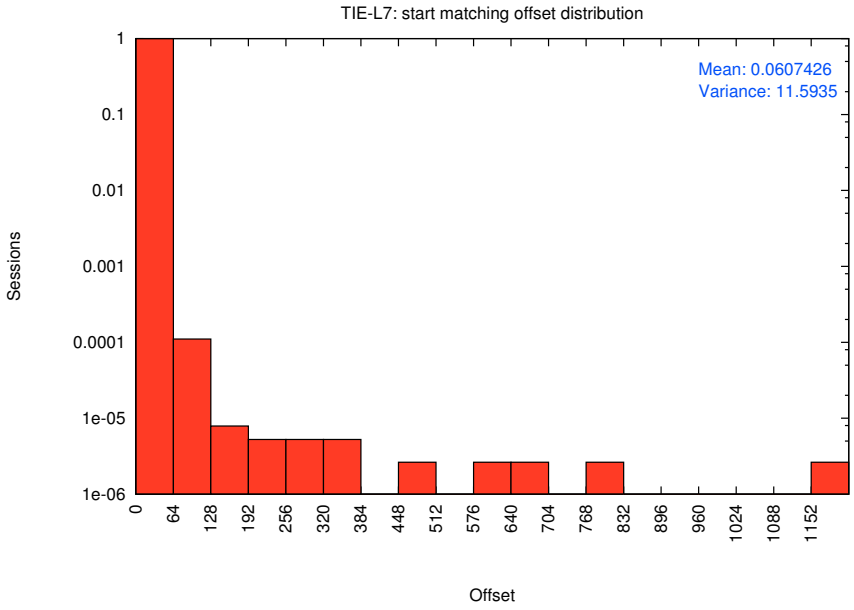


Figure 5.12: TIE-L7: Distribution of the offset of a matching string inside its first packet

5.4 Lightweight Payload inspection

Starting from the observations and findings shown in the previous section, here we propose and evaluate a different approach to payload inspection with the purpose of achieving significant improvements in terms of speed and resource requirements (i.e. traffic data, computational power, and memory) at the partial expense of classification accuracy. The motivation is to propose an easy-to-deploy lightweight payload inspection approach that can be used when privacy concerns and elevated bandwidth do not allow to run a deep payload inspection system. Examples are: to complement online classification systems based on machine-learning with a fast payload inspection technique, or to substitute minimal port-based classification in online contexts where an approximation of traffic breakdown is needed (e.g. continuously updated historical traffic graphs).

The findings from the previous section, together with our objectives brought us to define three main guidelines that led the design of the technique here proposed:

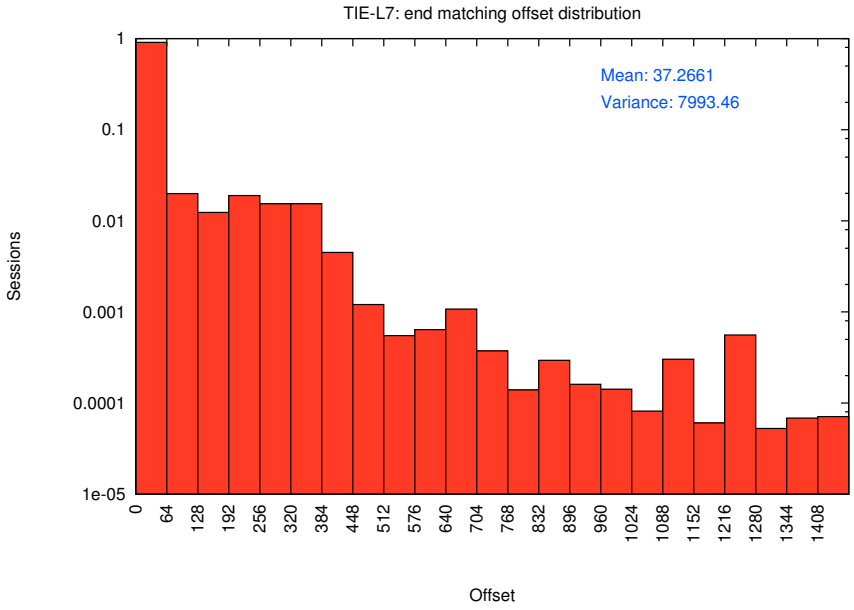


Figure 5.13: TIE-L7: Distribution of the offset of a matching string inside its last packet

- Using only the first packet for each direction.
- Using only the first few bytes of the considered packets.
- Not using pattern matching rules, but simple static string comparisons.

Even if relying only on the first few bytes and on the first payload-carrying packet may raise the objection that circumvention of this technique by protocol obfuscation would be easy, it is worth noting that protocol obfuscation techniques can be successful in general in circumventing all payload-inspection approaches based on string matching.

5.4.1 The N-Byte Classifier

To test the feasibility and evaluate the performance of a lightweight payload inspection approach we developed a TIE classification plugin that we named N-Bytes Classifier (TIE-NBC in the following). The TIE-NBC plugin implements a basic static string matching algorithm, applied to both directions of

the biflow under analysis, by accessing only the first few bytes of the first packet seen in each direction of a biflow.

Basically, the signature used by this algorithm is an association of a byte string to a *(application id, application subid)* pair. For each session (of type biflow), the first N bytes of payload carried by the first packet in the upstream direction are tested for matching against the list of signatures (N is specified at execution time as a command line option, however a maximum of 32 bytes is defined); the same is done for the downstream direction.

Signatures can be specific to upstream or downstream traffic, so the upstream packet is checked only against upstream signatures, and the same is done for the downstream one. As the directions sensed by TIE could be reversed with respect to the one considered by the signatures (some applications protocols are independent of direction), if the ordered match fails also a reversed direction attempt is made. Not all applications need the matching of both upstream and downstream flows: this can be specified on a per-signature basis. In this case the matching on one direction is enough to classify the session.

Moreover, it may happen that the sequence of known characters in the payload is prepended with, or interrupted by, a known number of variable characters. In order to represent the signature with a fixed string, a bitmask is associated to each signature. Each bit in the mask represents a character of the payload string: if the bit is 0 the relative character is to be ignored (can assume every value), if the bit is 1 then the character is compared against the next signature character. This leads to two advantages with respect to signature strings: they have the shortest format (showing just the known characters), and it is avoided the use of special meaning symbols to represent wildcard characters.

Few examples of signatures are reported in Figure 5.14. To make the bitmasks human readable, the constants (expressed in hexadecimal format) implementing the bitmasks are represented by preprocessor macros named with easy to read bit sequences. The bitmasks are represented as an `unsigned`

```
struct fingerprint udp[] = {       /* udp fingerprints collector */

        /* SNMP */
        { 0,     26,     true,    {snmp, snmp},    {SIZE(snmp), SIZE(snmp)}},
        /* DNS */
        { 0,     5,      true,    {dns, dns},    {SIZE(dns), SIZE(dns)}},
        /* DHCP */
        { 0,    9,       true,   {dhcp_up, dhcp_dw},    {SIZE(dhcp_up), SIZE(dhcp_dw)}},

        [...]
}

struct fingerprint tcp[] = {       /* tcp fingerprints collector */

        /* HTTP */
        { 0, 1, false, {http_up, http_dw}, {SIZE(http_up), SIZE(http_dw)} },
        /* POP */
        { 0,  18, false, {pop_up, pop_dw},  {SIZE(pop_up), SIZE(pop_dw)}},
        /* DNS */
        { 0,     5,      true,    {dns, dns},    {SIZE(dns), SIZE(dns)}},
        /* EDONKEY */
        { 0,  127, false, {edonkey_tcp_up, edonkey_tcp_dw}, {SIZE(edonkey_tcp_up),
                   SIZE(edonkey_tcp_dw)}},
        /* GNUtella */
        { 0, 128, false, {gnutella_tcp, gnutella_tcp}, {SIZE(gnutella_tcp),
                   SIZE(gnutella_tcp)}},

[...]
}
```

```
/* Arrays of signatures */

sig_str dns[]  = {{"\x00\x01\x00", M_111 >> 4}};

sig_str dhcp_up[]        = {{"\x01\x01\x06", M_111}};
sig_str dhcp_dw[]        = {{"\x02\x01\x06", M_111}};

sig_str edonkey_tcp_up[] = {{"\xe3\x00", M_101}};
sig_str edonkey_tcp_dw[] = {{"\xe3\x00", M_101}, {"\xc5\x00", M_101}};

[...]
```

```
/* Mask aliases: */
#define M_1 0x80000000
#define M_11 0xC0000000
#define M_101 0xA0000000
#define M_110010101011 0xCAB00000
#define M_111111100000001 0xFE020000
#define M_1100001111111111 0xC3FF0000
[...]
```

Figure 5.14: TIE-NBC: examples of signatures and masks

`integer`, 32 bit wide. This sets the limit of 32 bytes for the length of processable payloads.

```
struct fingerprint {
       u_int16_t sub_app_id;
       u_int16_t app_id;
       bool strict;
       sig_str *sig_arrays[2]; /* These are two arrays of sig_str .
                                  One for upstream, the other for downstream */
       u_int dim[2]; /* dimensions of the two arrays */
};

typedef struct {
       char *bytes;
       mask_t mask;
} sig_str;
```

Figure 5.15: TIE-NBC: main data structures used in the classify() function

The function that actually performs the string comparison between the payload and a signature is `sig_cmp()` (see Figure 5.16). By using a string comparison against a fixed single portion of the payload and without using any dynamic rule (as regular expression) the function, which could be also easily implementable in hardware, is very efficient and requires few computational resources. Moreover, the attempt is made only on the first packet seen for each direction of the biflow. In the next subsection we show an experimental analysis of the speed and computational requirements of TIE-NBC when compared against TIE-L7.

Signatures are grouped into a hierarchy that makes classification attempts fast (see Figures 5.14 and 5.15). At the highest level there are transport-level protocols. Each transport level protocol (e.g. TCP) has its own collector of *fingerprints* for the applications supported. An application fingerprint (e.g. for DNS over TCP) contains two arrays of signatures, respectively for the upstream and downstream directions. A signature consists into a (string, mask) pair.

This way, for each packet the matching process is done only on the appropriate subset of signatures, selected by transport protocol, then by direction (upstream/downstream). If a biflow signature is valid for both TCP and UDP transport protocols, then it is simply inserted into both collectors (e.g.

see `dns` signature, valid for both directions and both transport protocols, in Figure 5.14).

5.4.2 Experimental Evaluation

In this subsection we test TIE-NBC with a preliminary set of signatures and compare both its classification and computational performance against TIE-L7 and TIE-Port. Indeed, because TIE-NBC can be thought as a compromise between *(a)* speed, computational requirements, access to sensitive data, on a side, and *(b)* classification accuracy, on the other, we then can intuitively place it between port-based classification (the fastest technique) and deep payload inspection (the most accurate technique).

Performance of Classification

We first compare the classification results from TIE-NBC and TIE-Port (described in Chapter 4) against TIE-L7. We have collected TIE-NBC signatures for 61 different applications, with a total of 409 signatures (average of 6.7 signatures per application). The collection of the signatures has been done in three ways: (i) by manual adaptation from L7-filter signatures, (ii) by payload inspection and manual selection, (iii) adapting some signatures

```
bool sig_cmp(u_char* bytes, u_char bytes_len, sig_str *s)
{
        mask_t tmp_mask;
        u_int8_t i, j;
        u_int8_t sig_len = mask2len(s->mask);

        if (sig_len > bytes_len)
                return false;
        tmp_mask = 0x80000000; /* binary: 1000 ... 0000 */
        for (i = 0, j = 0; i < sig_len; i++) {
                if (TEST_BIT(s->mask, tmp_mask, 1)) {
                        if (bytes[i] != (u_char) s->bytes[j++])
                                return false;
                }
                tmp_mask >>= 1;
        }
        return true;
}
```

Figure 5.16: TIE-NBC: sig_cmp() function, performing string comparison with a byte mask.

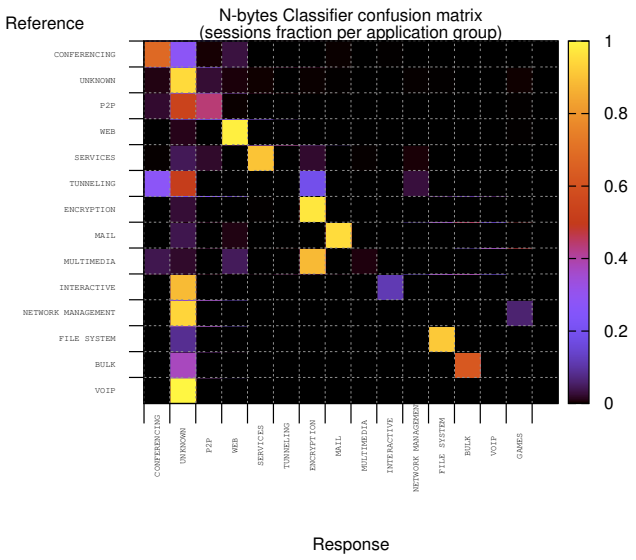


Figure 5.17: Confusion Matrix of TIE-NBC classification on the UNINA trace (TIE-L7 as ground truth)

used in [55] and listed at [143]. As regards adaptation of L7-filter signatures, at a first look they should be easy to translate in fixed strings, but this proved to be a non trivial task, due to a number of issues. The first one is due to the different kind of string matching algorithm: while TIE-L7 can match substrings everywhere in the flow, TIE-NBC by design does only per-character matching in their exact position, and only in the first packet. This leads to selection of a subset of available L7-filter signatures, namely the ones starting by "^" (which in regex syntax represents the start of the string). As TIE-L7 works on the whole payload stream of the biflow without discriminating upstream and downstream traffic (i.e. payloads are concatenated in a single stream following the order of arrival), regular expression rules representing signatures for the downstream direction do not start with "^". This led to the exclusion of signatures related to the downstream direction: indeed even if such signatures can be used as a hint in finding downstream-related signatures for TIE-NBC, this can not be easily automated. Moreover, wherever in L7-filter rules there is a "+" or "*" (allowing for sequence of characters

of arbitrary length), the following part of the regex can not be translated into a TIE-NBC signature. The translation criterion used in these cases is to expand the regex to one string and a number of associated bitmasks which hide an increasing number of ignored characters. This can be easily automated. We were able to separately add more rules by performing a manual selection upon a list of candidate strings collected by performing payload inspection, on pre-classified traffic, which reported common strings for each class ranked by occurrence. We then extracted and built some useful rules for TIE-NBC. After observing the promising results shown in the following, we foresee to make this process more elaborate and fully automated to (i) generate a large set of reliable signatures for TIE-NBC, (ii) add autonomous learning capabilities to TIE-NBC. The latter could be done by employing algorithms presented in literature for similar purposes (e.g. automated common substring extraction) [75] [89].

Table 5.6: Overall Accuracy of TIE-NBC and TIE-Port (TIE-L7 used as ground truth)

	Accuracy	Bytes-accuracy
TIE-NBC	66.0%	89.9%
TIE-Port	11.0%	13.5%

We performed an experimental evaluation conducted on several traces. Here we present the results obtained by processing the UNINA trace used also in Section 5.3 (about 50M packets, 400K biflows). To evaluate the expected loss in accuracy when moving from deep payload inspection to our lightweight approach, we used TIE-L7 as ground truth tool and then measured the classification accuracy of TIE-NBC. NBC reported an overall classification accuracy of 66% and an overall byte-accuracy of about 90%, showing a very good accuracy on *heavy* flows. Table 5.6 also reports overall classification accuracy obtained by TIE-Port, with (easily expectable) very low values. Figure 5.17 represents the confusion matrix (with applications grouped into categories) of TIE-NBC against TIE-L7. The *high* colors on the main diagonal testify a good accuracy on most applications, whereas few yellow cells outside of it help to identify the application categories that are

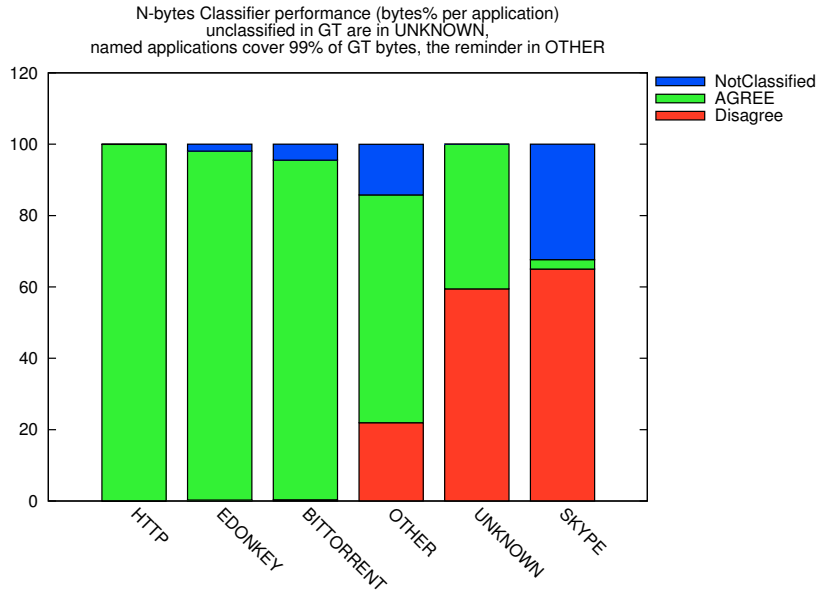


Figure 5.18: Comparing TIE-NBC classification of the UNINA trace against TIE-L7

not well identified by TIE-NBC. In Figure 5.18 we also summarized the classification results for the traffic classes associated to the largest byte-counts, where for each bar (corresponding to a class) it is illustrated the percentage of bytes on which TIE-NBC respectively agrees, disagrees, or returns *unknown*, with respect to TIE-L7. The figure shows that, among the top traffic classes, TIE-NBC has significant problems only with Skype. However, it must be noted, as will be observed also in Section 5.5, that the Skype rule of TIE-L7 is quite *aggressive*, generating several false positives. We also observe a characteristic that may indicate a positive property of TIE-NBC: about 40% of the traffic labeled as *unknown* by TIE-L7 is instead classified by TIE-NBC. Further investigastions are required to understand if in some situations the signatures from TIE-NBC can overcome TIE-L7, and there are also issues related to the reliability of the ground truth used, as reported in 5.5.

Classification time

TIE-NBC was conceived to gain processing speed and reduce resource usage with respect to deep payload inspection. To verify such expectations and

perform an experimental evaluation we performed a simple profiling of time and resource usage upon TIE-NBC, TIE-L7, and TIE-Port. We would like to highlight that a fair experimental comparison among three different classification techniques was in fact possible because all of them were implemented under the same platform (TIE). This allowed to introduce as less variability as possible when performing the different tests. For the same reasons, during the measurements, we verified that TIE was the only user process running on the host. None of the system related processes was CPU intensive or I/O intensive. The Operating System was GNU/Linux running the kernel linux 2.6.27-8-generic. Each measurement was performed by running TIE in *offline mode*, on the same traffic trace (UNINA), and by enabling each time only the classification plugin under analysis.

We define "classification time" as the difference between exit time from, and enter time in, the classify() function of the considered classification plugin (see Chapter 4). This function is called for each classification attempt: in TIE-Port and TIE-NBC this happens once per session, while in TIE-L7 it can be happen several times for each session (the default is a maximum of 10 times). This is taken into account in the analysis by considering both per-attempt and per-session statistics.

To measure the classification time, the code of the classification plugins has been instrumented with calls to the *gettimeofday()* POSIX function and logging of profiling information to a file. The resolution provided by the *gettimeofday()* function is of 1 μ *sec*. Accuracy actually depends on CPU clock frequency, but the measurements have been performed on a Intel(R) Pentium(R) 4 CPU 3.40GHz that has clock tick duration in the order of 1 *nsec*. Values measured for TIE-Port can be used as a reference for minimum attainable classification times. For TIE-Port, as shown in Figure 5.19, the classification time distribution is strongly concentrated in the first bin $[0, 10]\mu sec$, with about 95% of occurrencies. The rest drops down in less than $140\mu sec$. Sample mean value is $2.43\mu sec$ with sample variance of $1.10 * 10^{-11}$, on 447095 samples.

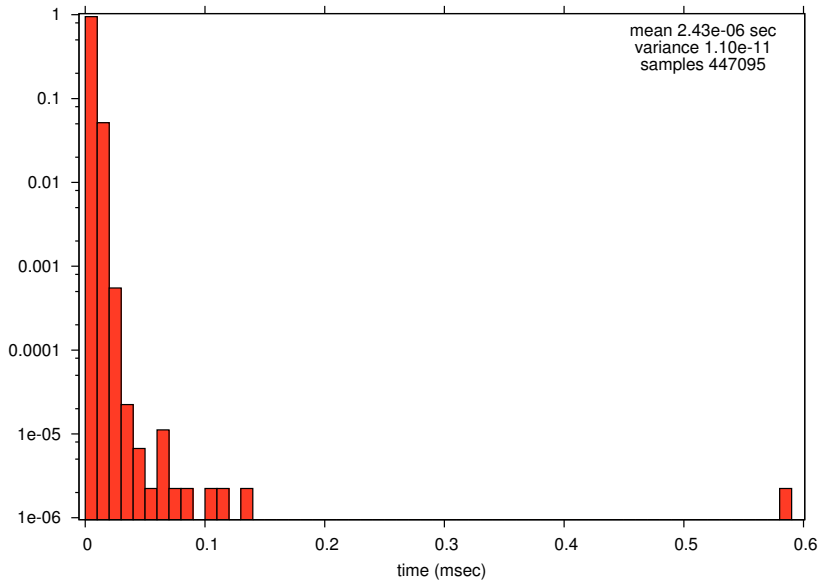


Figure 5.19: TIE-Port: frequency distribution of classification time. Only values above 10^{-6} are reported, in $10\mu sec$ wide bins.

For TIE-L7, as shown in Figure 5.20, we can find about 61% of classification attempts falling in the range $[100, 200]\mu sec$, about 14% in $[200, 300]\mu sec$, about 10% in $[0, 100]\mu sec$, with the rest of the samples spanned and rapidly decreasing around 3 msec. The sample mean value is $212\mu sec$ with sample variance of $8.04 * 10^{-8}$, on 770314 samples. This behavior, which is much different from the one showed by TIE-Port, was to be expected because of two main differences into the respective algorithms: the longer mean classification time of TIE-L7 is consistent with the use of regular expression rules opposed to hash table lookup; higher variance is due to differences in length and complexity of regular expressions that constitute TIE-L7 signatures, opposed to an almost constant time of hash table lookup performed by TIE-Port.

For TIE-NBC, as shown in Figure 5.21, we see a distribution similar to the one of TIE-Port. Most of the attempts fall in the first bin, counting about 74% in the range $[0, 10]\mu sec$, then about 19% in $[20, 30]\mu sec$, and the rest quickly decreasing to less than $170\mu sec$. The sample mean value is $9.30\mu sec$ with sample variance of $4.87 * 10^{-10}$, on 355499 samples. These

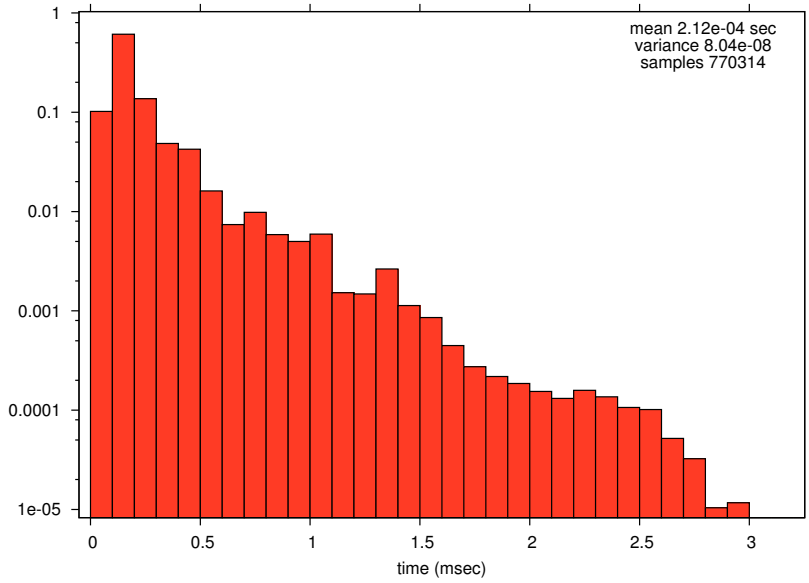


Figure 5.20: TIE-L7: frequency distribution of classification time. Only values above 10^{-5} are reported, in $100\mu sec$ wide bins.

results are consistent with the fact that the fixed string comparison done by TIE-NBC is much less computationally expensive than regular expression pattern matching, and that TIE-NBC signatures show small variability.

The overall results are summarized in Table 5.7, where we report both per-attempt and per-session values. This is because, as said, while TIE-Port and TIE-NBC process each session exactly once, TIE-L7 can try to classify several times each session. The mean value of classification attempts per session is obtained from the analysis reported in Section 5.3. From this comparison we

Table 5.7: Classification Time Comparison

Classifier	Attempts	per-session Mean Attempts	per-attempt Mean Time (μsec)	per-attempt Variance	per-session Mean Time (μsec)
TIE-Port	447095	1	2.43	$1.10 * 10^{-11}$	2.43
TIE-NBC	355499	1	9.30	$4.87 * 10^{-10}$	9.30
TIE-L7	770314	1.7	212	$8.04 * 10^{-8}$	360.4

find that the lightweight payload inspection technique implemented in TIE-NBC cuts mean classification time by 97.5% with respect to the deep payload inspection implemented by TIE-L7, reporting timings comparable with the

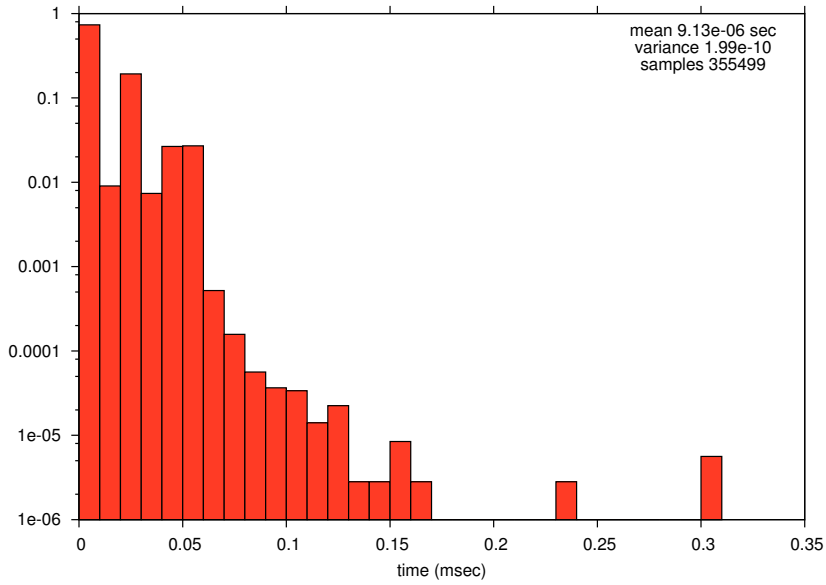


Figure 5.21: TIE-NBC: frequency distribution of classification times. Only values above 10^{-6} are reported, in $10\mu sec$ wide bins.

port-based classification performed by TIE-Port (while still retaining much of the classification accuracy of payload inspection).

Resource usage

In order to compare resource usage by TIE-L7 and TIE-NBC, memory usage and CPU time have been tracked during the execution of TIE with the respective classification plugins enabled. To have a ground level for comparison, also the port-based classifier TIE-Port has been tracked for memory and CPU usage. As for the previous tests, each classifier has been measured by running TIE on the same traffic trace (UNINA) and by enabling only the classification plugin under analysis. The system was always under conditions of low load. Using the system utility `ps`, values of *VSZ* and *CPU* are sampled with a period of 30 seconds, and logged to a file. An awk script processed these log files to extract statistics and a graph of usage-variations in time. VSZ is the *Virtual SiZe*, i.e. the total amount of memory used by the process for both code and data; it does not have to be totally stored in

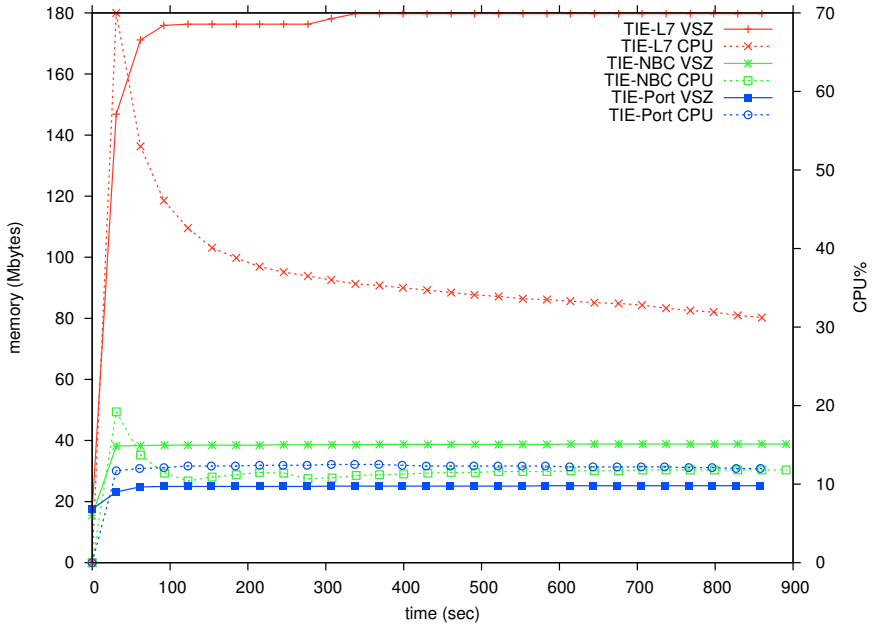


Figure 5.22: Memory and CPU usage over time

memory, but it can be partially swapped to disk as well. Besides obvious memory usage, the bigger VSZ is, the higher is the probability that a swap to disk is needed, with significant impact on overall performance. CPU is the percentage of time spent using the CPU (both for user and system code) with respect to the total running time of the process. It can be used as an index of CPU intensiveness of the technique under measure. Results of our measurements are reported in Figure 5.22.

Regarding CPU usage, TIE-L7 and TIE-NBC show similar qualitative behavior, reaching the maximum in the first 60 seconds, we then see a smooth decrease to a steady value. Port-based soon reaches its steady level. The slow decay is probably due to the fact that CPU% reported by `ps` acts by its definition as a moving average. TIE-Port reports a mean CPU usage of 12% (variance of 0.22), 12% for TIE-NBC (variance of 5.16) and 38% for TIE-L7 (variance 134).

Regarding memory usage, TIE-NBC behaves much like TIE-Port, reaching soon (into the first two 30seconds-separated samples) its steady value. TIE-L7 is slower in reaching its maximum. TIE-Port reports a mean memory

usage of 25500 KB (variance of 797360), 39496 KB for TIE-NBC (variance of 1247213) and 179166KB for TIE-L7 (variance 362043279). The almost constant behavior of memory usage in time for all the three classifiers can be explained with frequent activation of the TIE garbage collection system, triggered by the number of packet processed (roughly proportional to memory usage) (see Chapter 4).

TIE-NBC presents a reduction of 78% in memory usage with respect to TIE-L7. The much higher quantity of memory required by TIE-L7 is due to to the need to store more packets per session and more payload per packet.

5.4.3 Discussion

Starting from the empirical study of the *deepness* of payload inspection for traffic classification presented in Section 5.3, we proposed a different approach to payload inspection that we named *lightweight*. In this section we motivated and presented the proposed technique and its implementation in the TIE framework as a classification plugin (TIE-NBC). Moreover we conducted an experimental evaluation that has shown how TIE-NBC offers several advantages when compared to the deep payload inspection implemented by L7-filter and TIE-L7, at the expense of a partial loss of overall classification accuracy. We summarize and highlight here some of the main findings:

- **Traffic Data.** TIE-NBC needs access only to the first two packets per direction of a biflow and to a maximum of 32bytes only for each of those packets. TIE-L7 requires full packets payload and access to at least the first 10 packets of each biflow.

- **Classification Accuracy.** When evaluating TIE-NBC classification accuracy by using TIE-L7 as a ground-truth reference we observed a per-session overall accuracy of 66% and a per-byte overall accuracy around 90%. Percentages obtained by TIE-Port under the same conditions are respectively around 11% and 13%.

- **Classification Speed.** TIE-NBC experiences a 97.5% reduction in mean classification time compared to TIE-L7. When compared to TIE-Port, we observe average values 4 and 148 times higher for TIE-NBC and TIE-L7 respectively.

- **Memory Requirements.** Under our experiments TIE-NBC shows a 78% memory reduction when compared to TIE-L7. Moreover we report respectively for TIE-NBC and TIE-L7 values 1.5 times and 7.0 times higher than the memory footprint of TIE-Port.

- **CPU Usage.** TIE-NBC shows values of CPU usage comparable to TIE-Port and more than 3 times smaller than TIE-L7.

We conclude that the results obtained show how the presented technique can be considered valuable in several contexts, for example, as a replacement of port-based classification, when a reduced portion of traffic is available for classification, or as part of multi-classifier systems for online classification. Moreover, after having proved the benefits of the presented technique, we will now focus on designing automated signature collection procedures and on the development of payload signatures specifically designed for such technique. It must indeed be observed that the, yet interesting, classification performance reported in this study has been obtained by using a set of signatures mainly derived by signatures of other classifiers.

5.5 An Experimental Analysis of Ground Truth Tools

The purpose of this section is to contribute to shed light on the reliability of tools used for establishing the ground truth in traffic classification experiments. We evaluate and compare two of the most used tools in literature by running them on two traffic traces: one from year 2006, the other one from year 2008. The results show some limitations of the systems under analysis

and how their decisions can be contradictory when examining the same traffic. We also derive information useful for the design of better ground-truth tools. Moreover, we highlight that, to the best of our knowledge, this represents the first work in literature focused on the comparison of two systems for ground truth in traffic classification.

We already discussed the subject of ground truth in Chapter 3. From Table 3.1, which reports the approaches used in some of the best scientific papers about traffic classification presented in literature, we can see that they typically rely on payload inspection. Often a set of heuristics are also used [55], while usually manual and semi-automated analysis is necessary to spot relevant problems that, by altering the reference data, may affect the experimental analysis. A detailed study proposing an approach, based on both payload inspection and semi-automated techniques, to the accurate labeling of traffic has been presented by Moore et al. in [72]. This has been used, for example, in [96] to build ground-truth. In [97] instead, the authors used a commercial tool [98], while the open-source IDS Bro [81] was complemented with a set of signatures and used as ground-truth tool in [102]. In this section we consider:

- L7-filter, introduced in Chapter 4, considered the state of the art of publicly available tools for payload inspection, and often used in scientific papers to build ground truth [99] [127]. We actually use TIE running the TIE-L7 classification plugin, which works on the same signature set and is based on the same algorithm (see Section 4.9).

- Crl_pay, which is the ground-truth tool originally developed by Thomas Karagiannis et al., based on CAIDA's CoralReef[77] and used in several scientific papers [106] [53] [144].

The tools are similar in that they rely on payload inspection. However, while L7 filter presents a variegate set of regular expressions to perform pattern matching against packets payload, Crl_pay is more limited in the number of payload signatures but is very focused on Peer-to-Peer file-sharing appli-

cations and also uses ports, payload size, and a set of heuristics [140]. To perform the comparison shown in the following, we used the L7-filter implementation under TIE and we exploited the set of tools for post-processing that were developed in the TIE framework. However, we had to overcome some architectural and formal differences between TIE-L7 and Crl_pay. The latter classifies flows, whereas TIE-L7 operates on biflows. Moreover Crl_pay classifies flows into application categories (MAIL, GAMES, etc.) except for Peer-to-Peer file sharing, for which is able to discriminate among a comprehensive set of applications. Obviously the labels and IDs used by the tools were also different. All this, required the development of extra tools for the translation of Crl_pay output in the TIE output format, and to map its class IDs into TIE's application and group IDs.

We tested TIE-L7 and Crl_pay against the same traces described and used in Section 5.2, shortly named UNINA and KAIST. While in Section 5.2 we removed mice biflows (biflows with less than ten packets for both directions) before performing labeling and classification, for the evaluation of TIE-L7 and Crl_pay instead, we first ran both of them on the original unfiltered traces. In Figures 5.23 and 5.24 we report the traffic breakdown obtained respectively by Crl_pay and TIE-L7 on the UNINA trace. Aside from observing that the percentages of the various traffic categories identified by them do not match, we note that there is a non-negligible portion of traffic labeled as *Unknown*: more than 20% of biflows for Crl_pay and about 18% for TIE-L7. However, we observed that, as regards Crl_pay, a large number of *Unknown* biflows are made by a single or very few packets. Indeed, after removing mice biflows we obtained the results shown in the same figures and tagged as "no mice". We also note that the biflows labeled as *Unknown* by Crl_pay do not carry much traffic, being the byte-count of *Unknown* biflows almost negligible both before and after removing mice biflows. As regards TIE-L7 instead, Figure 5.24 shows that the portion of traffic labeled as *Unknown* is related to biflows with large byte counts and that removing mice flows does not bring an improvement. We found similar results for the KAIST trace:

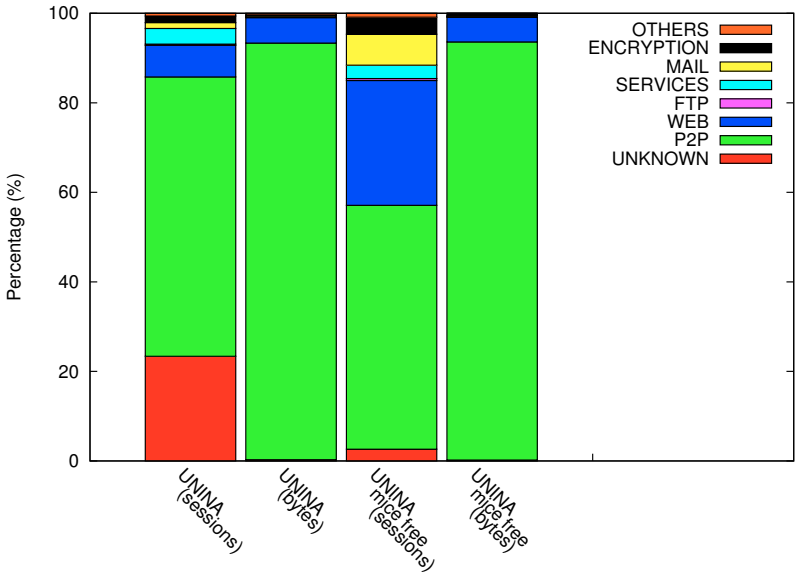


Figure 5.23: UNINA trace: Traffic breakdown performed by Crl_pay

about 10% of *Unknown* biflows for TIE-L7 and 8% for Crl_pay. The larger values for the UNINA trace can be explained by the date of the two traces: probably signatures from both tools need to be updated and perform better on an trace from 2006 (KAIST) when compared to one from 2008 (UNINA). However, we observed for the KAIST trace that while the percentage of unknown sessions gets quite lower after removing mice biflows, the remaining *Unknown* biflows this time still represent a large portion of the overall link traffic in terms of bytes (around 21GB on a total of 259GB). Behavior of TIE-L7 with respect to mice filtering, is instead approximately the same as the one observed for the UNINA trace.

We can conclude that both TIE-L7 and Crl_pay fail in labeling the entire traffic for both traces. This could be only partially explained by the presence of encrypted traffic. In the case of the KAIST trace the major hypothesis is the lack of signatures for some peer-to-peer file sharing applications, being that the Unknown traffic is represented by few flows but a large byte count. For the UNINA trace, being more recent, this could be due to not being able to keep such systems up to date with very recent traffic. Moreover we

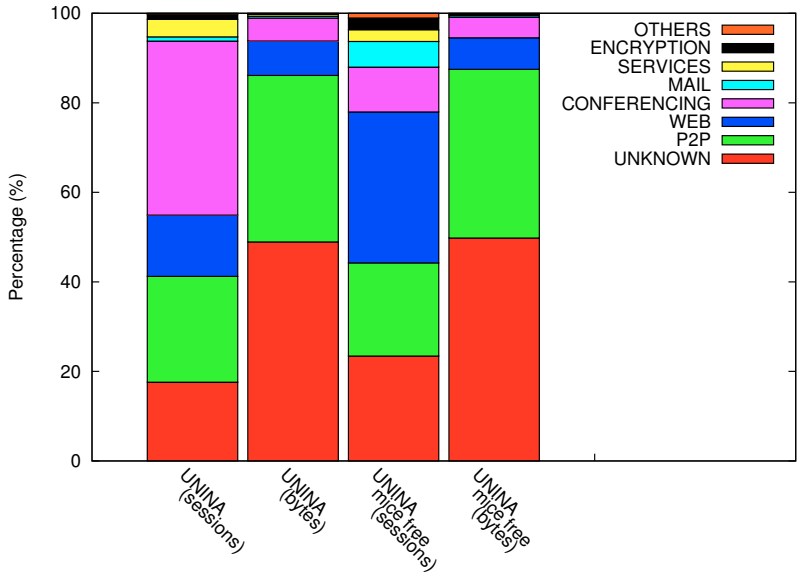


Figure 5.24: UNINA trace: Traffic breakdown performed by TIE-L7

observe that Crl_pay does not take into account Skype traffic (while TIE-L7 does), whose presence should be visible in a trace from year 2008 (we expect a more moderate presence of Skype traffic in the trace from year 2006).

After simply looking at the traffic breakdown reported by the two applications, we then started a detailed comparison of the groud-truth results obtained by TIE-L7 and Crl_pay on the UNINA trace, which is the most recent. We propose an approach to the comparison of two different ground-truth results that is based on a modified concept of confusion matrix. Usually a confusion matrix is built by considering a reference. Our problem here is that we are evaluating two systems commonly used for reference, and we do not know which one is wrong for each considered biflow (actually both could be). We therefore represent a comparison of the classification they performed by using two different confusion matrices: in the first one we show how one tool *misclassifies* the results obtained by the other, whereas in the second confusion matrix we do the opposite. In practice, we alternatively use one of the ground-truth tools as a reference. The matrices are normalized per row, as usual, but they are not square because TIE-L7 and Crl_pay identify a dif-

ferent number of classes. We therefore obtain a square sub-matrix to which columns associated to classes identified only by the second ground-truth tool are added.

By reading both confusion matrices we can understand when the two applications agree or not. In Figures 5.25 and 5.26 they are graphically represented. Whereas in Figures 5.27 and 5.28 we show confusion matrices calculated by grouping applications into categories. To easily obtain such data, related to tests on tens of thousands of samples, and to represent them graphically, we used the post-processing tools we designed for TIE's output files, taking advantage of the automated translation from the Crl_pay output format and application IDs that we have developed.

Observing such matrices reveals a lot of interesting information, which then must be analyzed case by case with manual intervention in order to be validated and understood. A lot of cells with *high* colors on the main diagonal of the square sub-matrix, however we can see that there are a lot of evident *misclassifications* (i.e. situations in which the two systems disagree).

The largely colored column labeled *Skype* in Figure 5.26 immediately shows that the Skype rules used by L7 filter are very aggressive. Indeed, we manually verified that they tend to generate false positives under several situations. This behavior is visible also in the by-group matrix in Figure 5.28 (Skype belongs to the "conferencing" category). However, Figure 5.25 tells us that a large part of the traffic recognized as *Skype* by TIE-L7 is labeled by Crl_pay as *Unknown*. This is consistent with Crl_pay not supporting Skype identification, while TIE-L7 seems to partially succeed in this. The confusion between *Skype*-labeled traffic by TIE-L7 and *Peer-to-Peer*-labeled traffic by Crl_pay instead, may be mainly due to two opposite situations we manually verified only partially: (i) Skype traffic generated by hosts running Peer-to-Peer file sharing applications is unknown to Crl_pay and is therefore catched by one of its heuristics, thus labelled as *P2P*; (ii) The Skype rules from TIE-L7 are very aggressive and can misclassify traffic that is actually *P2P* and correctly identified by Crl_pay which was specifically focused on

peer-to-peer traffic. As said, both situations may happen. In general we verified that Crl_pay has a good performance on Peer-to-Peer file sharing traffic, but sometimes its heuristics can lead to a chain-effect of erroneous results: let us say that host A exchanges Peer-to-Peer traffic with host B; host B then connects to host C using an application that is not recognized by Crl_pay; the result is that host C is then inserted into the table of hosts exchanging Peer-to-Peer file sharing traffic, and so on. While we observed that such heuristic can be very effective catching Peer-to-Peer traffic under most situations, we draw the conclusion that it can become very dangerous when the coverage of the payload signatures (or alternative techniques) used by the ground-truth application decreases.

The yellow-colored cells from the *Unknown* columns of the two matrices tell us for which applications the respective ground-truth system seems to systematically overcome the other one. E.g., all *VNC* and *SSDP* traffic identified by TIE-L7 (Figure 5.25) is labelled as *Unknown* by Crl_pay. Whereas all the games (*CRL_Games*) recognized by Crl_pay, but also *Earthstation* and *Peerenabler* (two peer-to-peer applications), are always labeled as *Unknown* by TIE-L7 (Figure 5.26). This happens also for most of the traffic identified as *IRC* by Crl_pay. Yellow-colored cells on the diagonal of the square sub-matrix (in both matrices), instead, show that the tools almost totally agree on several applications, as DNS, SSH, Direct Connect, and FTP. It is also interesting to observe from both kinds of matrices (e.g. Figures 5.27 and 5.28) how a lot of traffic considered *Unknown* by both systems overlaps. Indeed, around 40% of the biflows unclassified by TIE-L7 are also labeled as *Unknown* by Crl_pay, and viceversa. This confirms that there is a portion of traffic that is hard to classify with this kind of ground-truth approaches.

These are only few of the findings and hints that can be drawn by comparing the results of the two considered classifiers. As said, they always require manual investigation for a careful understanding, but the confusion matrices are, for example, very effective in spotting the most relevant areas of disagreement and inconsistencies. We started this study also with the purpose

to identify design criteria for a new ground-truth system, trying to take the best from both the applications analyzed. This will be implemented in TIE in the form of a set of separate classification plugins (Crl_pay techniques for Peer-to-Peer file sharing, heuristics, TIE-L7, etc.) that will be managed by a decision combiner specifically designed for performing off-line classification for ground truth.

The non-negligible percentage of unclassified traffic that sometimes both the considered tools report, should also raise an alert on the necessity to (i) be more rigorous in the evaluation and use of ground-truth tools, and (ii) to combine efforts towards the design and update of effective approaches. It is therefore our opinion that making public the tools used for ground-truth creation is even more important that making implementations of classification approaches available. Manual inspection of the traces is also an important duty of a researcher evaluating a traffic classification approach, but it can only be a complement to automated techniques when the number of flows considered ranges from tens of thousands to millions.

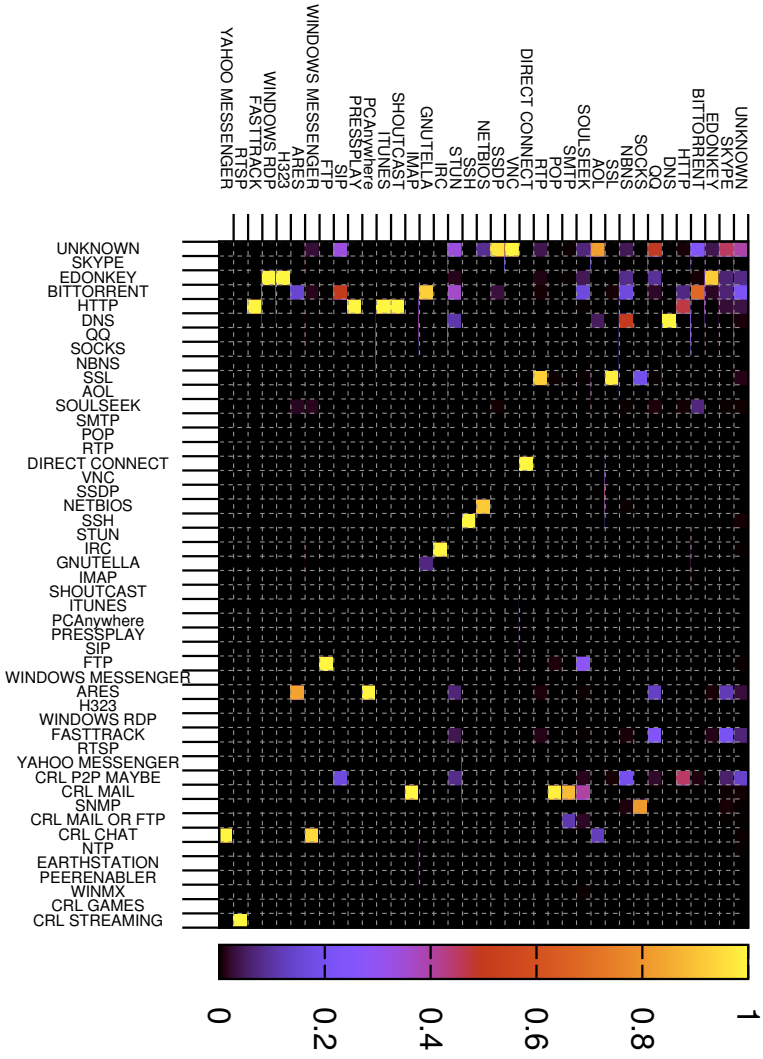


Figure 5.25: TIE-L7 *vs* Crl_pay comparison: per-application confusion matrix. (TIE-L7 used as ground truth)

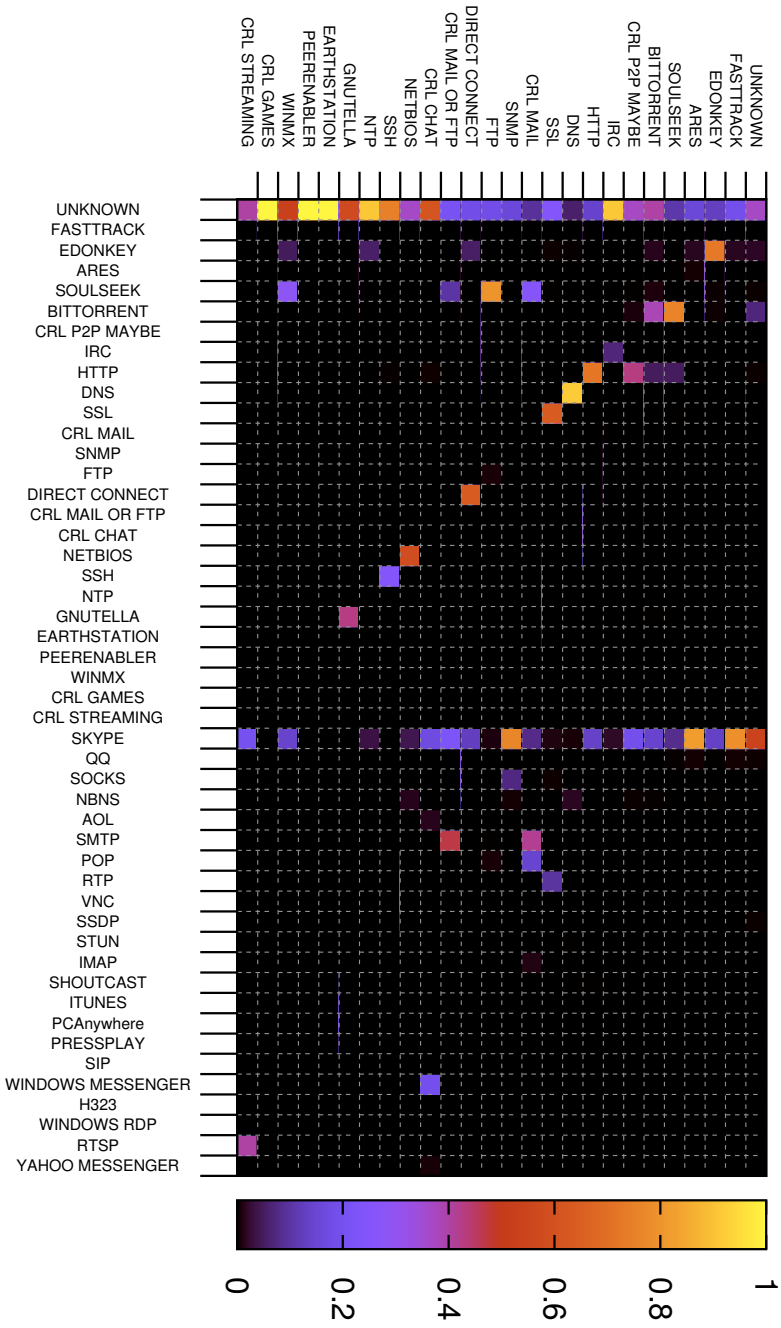


Figure 5.26: TIE-L7 *vs* Crl_pay comparison: per-application confusion matrix. (Crl_pay used as ground truth)

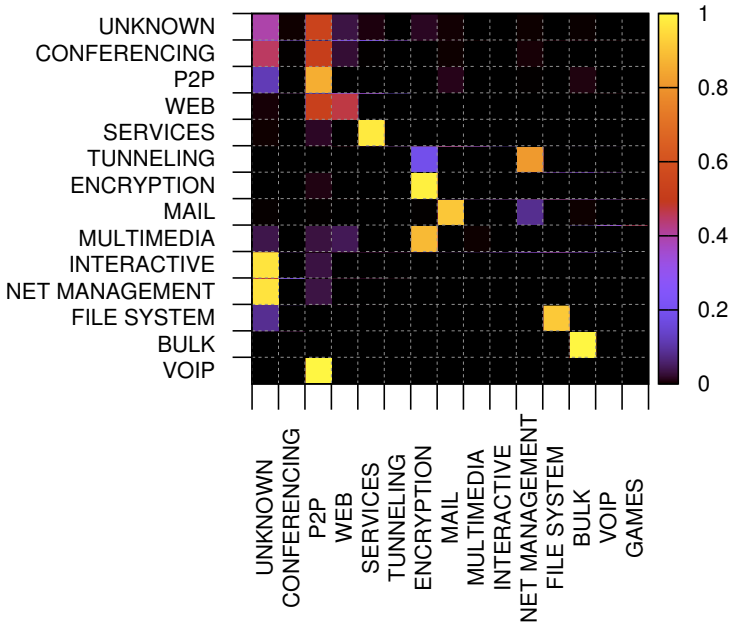


Figure 5.27: TIE-L7 *vs* Crl_pay comparison: per-category confusion matrix. (TIE-L7 used for ground truth)

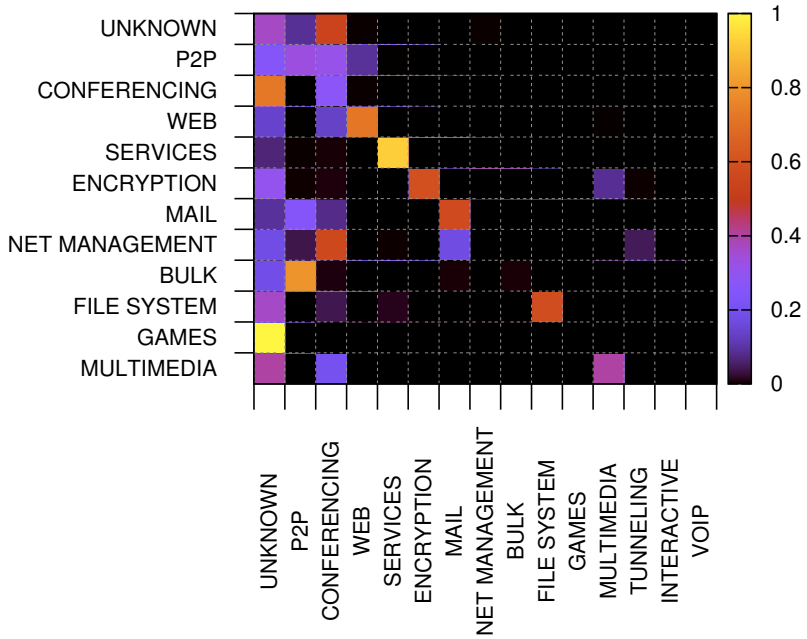


Figure 5.28: TIE-L7 *vs* Crl_pay comparison: per-category confusion matrix. (Crl_pay used for ground truth)

Chapter 6

Conclusions

Traffic classification is an important task with multiple applications in several fields of networking, from establishing knowledge on network usage to real-time enforcement of filtering rules on specific traffic categories. Because of its possible applications, traffic classification has several social and economical implications. However, despite its importance and the numerous works presented in literature, today the scenario is quite uncertain. Moreover, the few techniques currently available in practice show limited performance, which in the future are expected to further decrease because of current traffic trends (encryption, encapsulation, etc.).

It must also be considered that, because of conflicting interests related to traffic classification (e.g. providers want to classify and control traffic, while users often do not want to be controlled), users and software developers will constantly look for ways to evade the newly found classification techniques, opening an eternal challenge as it happened in other fields of networking (e.g. intrusion detection). A negative example may be malicious users circumventing security policies, but a totally different case may be users trying to avoid censorship or illegitimate control. The debate on letting providers access user data carried by packets (required by deep payload inspection techniques) is also a hot topic involving the scientific community. Discussing the role of science is outside the scope of this thesis, but from a general point of view science is in charge of solving problems and increasing knowledge of

phenomena.

Significant contributions from the scientific community, in regards to traffic classification and problems connected to it, should be not only to conceive new techniques and solutions, but also to rigorously evaluate them, in order to provide certain answers to questions regarding what can be done and by which means. For example, to which extent is it possible to classify network traffic without accessing packet payload? Is access to full payload data necessary to perform traffic classification through payload inspection?

This thesis shows instead that the large amount of work that has been proposed by the research community is difficult to evaluate and compare and that no actual implementations of the novel techniques recently investigated are currently made available. The first contribution of this work indeed, is a detailed and critical analysis of the literature and the state of art, from which we draw a precise list of open problems of research in traffic classification. Recommendations and considerations on the main issues to be faced by research in this field are identified and are the main motivations for the other contributions presented in this thesis.

We present a software tool made available to the scientific community that we designed to allow on-the-field evaluation and comparison of actual implementations of traffic classification techniques. We targeted it towards emerging problems like *online* traffic classification and combination of multiple techniques (multi-classifier systems). While we detail several design choices made with these intents, we think TIE's potential is partially demonstrated by its application in some of the experimental contributions presented in this thesis and in the interest in it that the scientific community has already shown.

The development of TIE finds its roots in significant work done in the theoretical and experimental analysis of network traffic that we carried out, which was also assisted by the development of specific software tools. Traffic classification can be indeed partially considered as a sub-field of traffic analysis, borrowing knowledge and techniques from it. Moreover, we can-

not design effective techniques to classify current network traffic if we do not understand properties of traffic generated by different applications, especially the emerging ones. For this reason, and to show the background behind the novel classification features that we later present, in this thesis we show several experimental results from the analysis of traffic generated by various applications. One of the main findings of these studies is that traffic observed at packet-level (i.e. in terms of packet size and inter-packet time) has statistical properties that look different depending on the network application considered and that are partially invariant with respect to space and time. Aside from exploiting such properties for statistical modeling and other purposes, we show that they can be very effectively applied to the problem of traffic classification. Another contribution of this thesis is indeed the proposal and evaluation of a traffic classification technique based on traffic features that have never been used before. Their discriminative power looks particularly interesting when considering that such features are expected to be quite robust to evasion attempts. This is because they are based on real observations of traffic from different applications rather than obtained by mechanically selecting features from a set of most properties describing traffic flows. Moreover, they do not require access to payload or to specific header fields used only by some applications (e.g. TCP flags). The analysis of literature here presented has indeed shown that we still need to search for more techniques alternative to payload inspection, which will probably need to be combined to maximize classification performance. At the same time we noted that all the practical implementations of real traffic classifiers today available are based on access to packet content. For this reason, we study this subject in terms of its currently-known limitations and accuracy. Firstly, we investigate what we call the *deepness* of deep payload inspection approaches (that is, the amount of data from packets' content that is practically used in order to successfully identify applications). Secondly, we experimentally evaluate classification accuracy of two *ground-truth* systems based on payload inspection. As for the first contribution, our study shows that most

successful matches obtained on a deep payload inspection tool for classification (considered the state of art of open tools) happen on the first packets exchanged and at the first bytes of payload. Based on this observation we propose a novel approach to payload inspection, defined *lightweight*, in the attempt to remove some of the limitations of current techniques. The experimental evaluation in this thesis shows that, at the expense of a partial reduction of accuracy, it is possible to reduce the computational load and to increase classification speed to values much closer to port-based classification (the fastest but least accurate technique by far) rather than to deep payload inspection. These results suggest several applications of the proposed classifier, for example: (i) in situations where port-based classification is today still considered the only practical choice (e.g. realtime web reports on link traffic); (ii) in the context of *online* classification through multi-classifier systems (a lightweight and more privacy-friendly payload inspection approach can indeed be combined to different techniques with the purpose to develop accurate *online* traffic classifiers). Finally, we study the problem of *ground truth*. To the best of our knowledge for the first time in literature, we question how accurate tools used for establishing ground truth in traffic classification could be. We perform an experimental evaluation revealing serious contradictions and inability to exhaustively classify traffic from a network link. The contribution of this last study is two-fold. On one side it shows that accuracy of payload-based approaches is diminishing, motivating the search for alternative solutions or their combination with different approaches. On the other side, it confirms some of the issues that we highlighted when analyzing the state of art: the dubious accuracy of ground-truth systems and the lack of rigorous and complete evaluation of the new classification techniques there are presented by the scientific community.

Concluding, research in traffic classification is prolific and based on strong motivations, however there are still many questions to be rigorously answered, especially when considering the continuously evolving scenario of network traffic and of the Internet in general. In order to improve the state

of art in traffic classification, this thesis contributes with strategic tools, new techniques, and by identifying issues and future directions.

Bibliography

[1] Johannes Färber. Network game traffic modelling. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 53–57, New York, NY, USA, 2002. ACM.

[2] S. O. Bradner K. Claffy, S.D. Meinrath. The (un)economic internet? *Internet Computing, IEEE*, 11(3):53–58, May-June 2007.

[3] V. Paxson S. Floyd. Difficulties in simulating the internet. *IEEE/ACM Trans. Networking*, 9(4):392–403, August 2001.

[4] A. Pescapé A. Dainotti. Plab: a packet capture and analysis software architecture. *Dip. di Informatica e Sistemistica, Univ. of Naples Federico II, Italy. Tech. Rep. TR-DIS-122004*, October 2004. http://www.grid.unina.it/Traffic/pub/TR-DIS-122004.pdf.

[5] V. Jacobson S. McCanne. The bsd packet filter: A new architecture for userlevel packet capture. *Winter 1993 USENIX Conference*, pages 259–269, January 1993.

[6] *Traffic Project*. http://www.grid.unina.it/Traffic/.

[7] S. Jamin D. Mitzel R. Caceres, P. Danzig. Characteristics of wide-area tcp/ip conversations. *ACM SIGCOMM Computer Communication Review*, 21(4):101–112, September 1991.

[8] R. Caceres D. Mitzel P. Danzig, S. Jamin and D. Estrin. An empirical workload model for driving wide-area tcp/ip network simulations. *Internetworking: Research and Experience*, 3(1):1–26, March 1992.

[9] A. Bestavros M. Crovella. Self-similarity in world-wide web: Evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, December 1997.

[10] W. Willinger A. Erramilli, O. Narayan. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, April 1996.

[11] B. A. Mah. An empirical model of http network traffic. *INFOCOM '97*, 2:592–600, April 1997.

[12] B. Ahlgren H. Abrahamsson. Using empirical distributions to characterize web client traffic and to generate synthetic traffic. *GLOBECOM '00*, 1:428.433, November 2000.

[13] Y. Gao K. Jeffay F.D. Smith M.C. Weigle J. Cao, W.S. Cleveland. Stochastic models for generating synthetic http source traffic. *INFOCOM 2004*, 3:1546–1557, March 2004.

[14] M. Howarth L. Liang, Z. Sun. Measurement and modelling of www traffic in a lan environment. *EUROCON 2003*, 1:22–24, September 2003.

[15] E. Chlebus R. Ohri. Measurement based e-mail traffic characterization. *SPECTS'05*, July 2005.

[16] G. A. Marin S. Luo. Realistic internet traffic simulation through mixture modeling and case study. *2005 Winter Simulation Conference*, pages 2408–2416, 2005.

[17] k. Claffy S. McCreary. Trends in wide area ip traffic patterns: A view from ames internet exchange. *13th ITC Specialist Seminar on Measurement and Modeling of IP Traffic*, September 2000.

[18] Steven Gargolinski, Christopher St. Pierre, and Mark Claypool. Game server selection for multiple players. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–6, New York, NY, USA, 2005. ACM.

[19] M. S. Borella. Source models of network game traffic. *Computer Communications*, 23(4):403–410, February 2000.

[20] Wu chang Feng, Francis Chang, Wu chi Feng, and Jonathan Walpole. Provisioning on-line games: A traffic analysis of a busy counter-strike server. In *in Proc. of the Internet Measurement Workshop*, pages 151–156, 2002.

[21] David LaPoint Mark Claypool and Josh Winslow. Network analysis of counter-strike and starcraft. In *In Proceedings of the 22nd IEEE International Performance, Computing, and Communications Conference (IPCCC)*, April 2003.

[22] P. Branch H. Choo T. Lang, G.J. Armitage. A synthetic traffic model for half-life. In *Australian Telecommunications Networks & Applications Conference (ATNAC 2003)*, December 2003.

[23] V. K. Prabhu C. Heyaime-Duvergé. Modeling action and strategy internet-games traffic. In *IEEE VTC 2002*.

[24] *Counter Strike*. http://www.counter-strike.net.

[25] M. Meo D. Ciullo, M. Mellia. Traditional ip measurements: What changes in a today multimedia ip network. In *Proc. of Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008*, February 2008.

[26] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs, and Hui Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 107–120, New York, NY, USA, 2004. ACM.

[27] B. Li X. Zhang, J. Liu. On large-scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental results. In *Proc. MMSP, 2005*.

[28] B. Li T. P. Yum X. Zhang, J. Liu. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE Infocom*, 2005.

[29] J. Liang Y. Liu K. W. Ross X. Hei, C. Liang. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[30] J. Liang Y. Liu K. W. Ross X. Hei, C. Liang. A measurement study of a large-scale p2p iptv system. In *in IEEE Transactions on Multimedia*, December 2007.

[31] K. W. Ross X. Hei, Y. Liu. Inferring network-wide quality in p2p live streaming systems. In *IEEE JSAC, special issue on P2P Streaming*, December 2007.

[32] J. Liang K. Nahrstedt L. Vu, I. Gupta. Measurement and modeling of a large-scale overlay for multimedia streaming. In *Proc. of QSHINE'07, International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, December 2007.

[33] H. Zhang S. Ali, A. Mathur. Measurement of commercial peer-to-peer live video streaming. In *Proc. of Workshop in Recent Advances in Peer-to-Peer Streaming*, 2006.

[34] A. Botta A. Dainotti A. Pescapé G. Ventre K. Salamatian T. Silverston, O. Fourmaux. Traffic analysis of peer-to-peer iptv communities. *Computer Networks*, In Press.

[35] V. Paxson S. Staniford and N. Weaver. How to own the internet in your spare time. *11th USENIX Security Symposium*, 2002.

[36] D. Moore C. Shannon. The spread of the code-red worm. http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml.

[37] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[38] D. Moore C. Shannon. The spread of the witty worm. *IEEE Security & Privacy*, 2(4):46–50, July 2004.

[39] S. Staniford R. Cunningham N. Weaver, V. Paxson. A taxonomy of computer worms. *ACM CCS First Workshop on Rapid Malcode (WORM)*, October 2003.

[40] MIT LCS Robert Beverly. Ms-sql slammer/sapphire traffic analysis. 2003. http://momo.lcs.mit.edu/slammer/.

[41] D. Towsley C. C. Zou, W. Gong. Code red worm propagation modeling and analysis. *9th ACM CCCS*, November 2002.

[42] K. Kwiat Z. Chen, L. Gao. Modeling the spread of active worms. *IEEE INFOCOM 2003*, 2003.

[43] E.P. Markatos P. Akritidis, K. Anagnostakis. Efficient content-based detection of zero-day worms. *ICC 2005*, May 2005.

[44] J. Jung S. Schechter and A. W. Berger. Fast detection of scanning worm infections. *7th RAID*, September 2004.

[45] S. Staniford N. Weaver and V. Paxson. Very fast containment of scanning worms. *USENIX Security Symposium*, August 2004.

[46] D. Towsley L. Gao C.C. Zou, W. Gong. The monitoring and early detection of internet worms. *IEEE/ACM Trans. on Networking*, 13(5):961–974, October 2005.

[47] S. Vander Wiel T. Woo T. Bu, A. Chen. Design and evaluation of a fast and robust worm detection algorithm. *IEEE INFOCOM 2006*.

[48] G. Ventre A. Dainotti, A. Pescapè. Worm traffic analysis and characterization. In *IEEE ICC 2007*, June 2007.

[49] *MAWI (Measurement and Analysis on the WIDE Internet) Working Group*. http://www.wide.ad.jp/wg/mawi/.

[50] G. Ventre A. Dainotti, A. Pescapé. A packet-level characterization of network traffic. *CAMAD 2006*, 2006.

[51] K. Jeffay F. D. Smith, F. H. Campos and D. Ott. What tcp/ip procotol headers can tell us about the web. *ACM SIGMETRICS 2001*, pages 245–256, 2001.

[52] B. Lyles C. Cotton M. Khan D. Moll R. Rockell T. Seely C. Diot C. Fraleigh, S. Moon. Packet-level traffic measurements from the sprint ip backbone. *IEEE Network*, 17(6):6–16, December 2003.

[53] Thomas Karagiannis, Andre Broido, Nevil Brownlee, KC Claffy, and Michalis Faloutsos. Is p2p dying or just hiding? In *IEEE Globecom*, 2004.

[54] Cisco Systems. *Blocking Peer-to-Peer File Sharing Programs with the PIX Firewall*. http://www.cisco.com/application/pdf/paws/42700/block_p2p_pix.pdf.

[55] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and kc claffy. Transport layer identification of p2p traffic. In *ACM IMC*, October 2004.

[56] *mshmro.com*. http://www.mshmro.com.

[57] *myp2p.eu*. http://myp2p.eu [April 2008].

[58] Yan Huang, Tom Z.J. Fu, Dah-Ming Chiu, John C.S. Lui, and Cheng Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 375–388, New York, NY, USA, 2008. ACM.

[59] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA*, May 2003.

[60] *Peer-to-Peer TV traffic traces.* http://content.lip6.fr/traces [April 2008].

[61] *Windump.* http://windump.polito.it [April 2008].

[62] *Plab.* http://www.grid.unina.it/software/Plab [April 2008].

[63] *Joost.* http://www.joost.com [April 2008].

[64] The CAIDA Dataset on the Witty Worm - March 19-24, 2004, Colleen Shannon and David Moore, http://www.caida.org/passive/witty/. Support for the Witty Worm Dataset and the UCSD Network Telescope are provided by Cisco Systems, Limelight Networks, the US Department of Homeland Security, the National Science Foundation, and CAIDA, DARPA, Digital Envoy, and CAIDA Members.

[65] Jeffrey Erman, Martin Arlitt, Anirban Mahanti, and Carey Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *WWW*, May 2007.

[66] J. Kurose D. Towsley K. Suh, D.R. Figueiredo. Characterizing and detecting skype-relayed traffic. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, April 2006.

[67] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, New York, NY, USA, 2007. ACM.

[68] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking down skype traffic. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 261–265, 2008.

[69] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *ACM IMC*, October 2004.

[70] Andrew Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS*, June 2005.

[71] IANA. *IANA Port Numbers*. http://www.iana.org/assignments/port-numbers.

[72] Andrew Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *PAM*, April 2005.

[73] HT Ju MS Kim JW Hong YJ Won, BC Park. A hybrid approach for accurate application traffic identification. *IEEE/IFIP E2EMON*, 2006.

[74] D. P. Olshefski D. Saha Zon-Yin Shae C. Waters R. B. Jennings, E. M. Nahum. A study of internet instant messaging and chat protocols. *IEEE Network*, 20(4):16–21, July-August 2006.

[75] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: Automataed construction of applicatin signatures. In *SIGCOMM MineNet Workshop*, August 2005.

[76] F. Gringoli L. Salgarelli M. Dusi, M. Crotti. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Elsevier Computer Networks (COMNET)*, in print (2008).

[77] *CoralReef*. http://www.caida.org/tools/measurement/coralreef/.

[78] Yin Zhang and Vern Paxson. Detecting backdoors. In *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, pages 12–12, Berkeley, CA, USA, 2000. USENIX Association.

[79] *Regular Expressions*. The Single UNIX Specification, Version 2, The Open Group, 1997.

[80] *L7-filter, Application Layer Packet Classifier for Linux*. http://l7-filter.sourceforge.net.

[81] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, pages 23–24, 1999.

[82] M. Morandi O. Baldini A. Monclus P. Risso, F. Baldi. Lightweight, payload-based traffic classification: An experimental evaluation. *IEEE International Conference on Communications, 2008 (ICC '08)*, May 2008.

[83] Andrew R. Baker, Brian Caswell, and Mike Poor. *Snort 2.1 Intrusion Detection - Second Edition*. Syngress, 2004.

[84] Jose M. Gonzalez, Vern Paxson, and Nicholas Weaver. Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 139–149, New York, NY, USA, 2007. ACM.

[85] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 7:2745–2769, December 2006.

[86] Taesang Choi, Changhoon Kim, Seunghyun Yoon, Jeongsook Park, Byungjun Lee, Hyunghan Kim, and Hyungseok Chung. Content-aware internet application traffic measurement and analysis. In *IEEE/IFIP NOMS*, April 2004.

[87] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, May 2004.

[88] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *ACM IMC*, 2006.

[89] Byung-Chul Park, Young J. Won, Myung-Sup Kim, and James W. Hong. Towards automated application signature generation for traffic identification. In *IEEE NOMS*, April 2008.

[90] *Juniper Networks*. http://www.juniper.net.

[91] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical Report RR-05-13, Department of Computer Science, Queen Mary, University of London, 2005.

[92] Charles Wright, Fabian Monrose, and Gerald M. Masson. Hmm profiles for network traffic classification. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 9–15, New York, NY, USA, 2004. ACM.

[93] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Internet traffic identification using machine learning. In *IEEE Globecom*, November 2006.

[94] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *IEEE LCN*, November 2005.

[95] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classificaton using clustering algorithms. In *ACM SIGCOMM MineNet Workshop*, September 2006.

[96] Tom Auld, Andrew W. Moore, and Stephen F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, January 2007.

[97] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *ACM CoNEXT*, December 2006.

[98] Qosmos. http://www.qosmos.com.

[99] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM CCR*, 37(1):7–16, January 2007.

[100] Wei Li and Andrew W. Moore. A machine learning approach for efficient traffic classification. *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2007.

[101] *WEKA: Data Mining Software in Java.* http://www.cs.waikato.ac.nz/ml/weka/.

[102] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. In *IFIP Performance*, October 2007.

[103] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68, New York, NY, USA, 2004. ACM.

[104] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, to appear, 2008.

[105] S. Tafvelin W. John. Heuristics to classify internet backbone traffic based on connection patterns. *International Conference on Information Networking, 2008 (ICOIN 2008)*, January 2008.

[106] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. In *ACM SIGCOMM*, August 2005.

[107] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs. In *ACM IMC*, October 2007.

[108] Marcell Perényi and Sándor Molnár. Enhanced skype traffic identification. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[109] Dario Rossi, Silvio Valenti, Paolo Veglia, Dario Bonfiglio, Marco Mellia, and Michela Meo. Pictures from the skype. *SIGMETRICS Perform. Eval. Rev.*, 36(2):83–86, 2008.

[110] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. In *PAM*, pages 165–175, 2007.

[111] A. Nur Zincir-Heywood Riyad Alshammari. Investigating two different approaches for encrypted traffic classification. *Sixth Annual Conference on Privacy, Security and Trust, 2008 (PST '08)*, pages 156–166, October 2008.

[112] *netAI: Network Traffic based Application Identification.* http://caia.swin.edu.au/urp/dstc/netai.

[113] *Tstat.* http://tstat.tlc.polito.it [November 2008].

[114] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Self-learning ip traffic classification based on statistical flow characteristics. In *PAM*, 2005.

[115] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM CCR*, 36(5):7–15, October 2006.

[116] Luca Salgarelli, Francesco Gringoli, and Thomas Karagiannis. Comparing traffic classifiers. *SIGCOMM Comput. Commun. Rev.*, 37(3):65–68, 2007.

[117] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Byte me: A case for byte accuracy in traffic classification. In *ACM SIGMETRICS MineNet Workshop*, June 2007.

[118] M. Steinbach P.N. Tan and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc., Boston Ma, USA, 2005.

[119] F. Gringoli L. Salgarelli C. Sansone A. Este, F. Gargiulo. Pattern recognition approaches for classifying ip flows. *The 7th International Workshop on Statistical Pattern Recognition*, December 2008.

[120] Tim Strayer, Mark Allman, Grenville Armitage, Steve Bellovin, Shudong Jin, and Andrew W. Moore. Imrg workshop on application classification and identification report. *SIGCOMM Comput. Commun. Rev.*, 38(3):87–90, 2008.

[121] *COST Action IC0703: Data Traffic Monitoring and Analysis (TMA): theory, techniques, tools and applications for the future networks*. 2nd meeting, Samos Island (Greece), 22-23 Sept. 2008.

[122] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[123] Tim Bass. Intrusion detection systems and multisensor data fusion. *Commun. ACM*, 43(4):99–105, 2000.

[124] Giorgio Giacinto, Fabio Roli, and Luca Didaci. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern Recogn. Lett.*, 24(12):1795–1803, 2003.

[125] *Tcpdump and the Libpcap library*. http://www.tcpdump.org [November 2008].

[126] Thomas H. Ptacek, Timothy N. Newsham, and Homer J. Simpson. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, 1998.

[127] Zhu Li, Ruixi Yuan, and Xiaohong Guan. Accurate classification of the internet traffic based on the svm method. In *ICC*, June 2007.

[128] *Tcpreplay*. http://tcpreplay.sourceforge.net [November 2008].

[129] *Netfilter/IPTables*. http://www.netfilter.org [November 2008].

[130] A. Pescapè P. Salvo Rossi A. Dainotti, W. De Donato. Classification of network traffic via packet-level hidden markov models. In *IEEE GLOBECOM 2008*, December 2008.

[131] *RECIPE (Robust and Efficient traffic Classification in IP nEtworks)*. http://recipe.dis.unina.it.

[132] *NETQOS - Policy Based Management of Heterogeneous Networks for Guaranteed QoS*. http://www.netqos.eu.

[133] *COST Action IC0703: Data Traffic Monitoring and Analysis (TMA): theory, techniques, tools and applications for the future networks*. http://www.cost-tma.eu.

[134] Alberto Dainotti, Alessio Botta, Antonio Pescapé, and Giorgio Ventre. Searching for invariants in network games traffic. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–2, New York, NY, USA, 2006. ACM.

[135] Anothony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *PAM*, April 2004.

[136] Thuy T.T. Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *IEEE LCN*, November 2006.

[137] Nigel Williams, Sebastian Zander, and Grenville Armitage. Evaluating machine learning algorithms for automated network application identification. Technical Report 060401B, CAIA, Swinburne Univ., April 2006.

[138] Kristin Bennett and Colin Campbell. Support vector machines: Hype or hallelujah? *ACM SIGKDD Explorations*, 2(2):1–13, 2000.

[139] John C. Plat. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[140] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. Technical report, University of California Riverside, 2005.

[141] Young J. Won, Byung-Chul Park, Hong-Taek Ju, Myung-Sup Kim, and James W. Hong. A hybrid approach for accurate application traffic idenficiation. In *IEEE/IFIP E2EMON*, April 2006.

[142] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed.* Morgan Kaufmann, 2005.

[143] Thomas Karagiannis. *Application-specific Payload Bit Strings.* http://www.cs.ucr.edu/~tkarag/papers/strings.txt, 2004.

[144] Hyunchul Kim, Marina Fomenkov, kc claffy, Nevil Brownlee, Dhiman Barman, and Michalis Faloutsos. Comparison of internet traffic classification tools. In *IMRG Workshop on Application Classification and Identification*, October 2007.