POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Ingegneria Elettronica e delle Communicazioni –
XXIV ciclo

Tesi di Dottorato

Analysis, characterization and classification of Internet traffic

Alessandro Finamore

Tutore
prof. Michela Meo

Coordinatore del corso di dottorato
prof. Ivo Montrosset

Febbraio 2012

Acknowledgements

A Fabio, per avermi introdotto nel mondo delle reti e a Marco, Michela e Maurizio, per il constante supporto, la guida professionale ma soprattutto per l'amicizia che hanno reso questi anni una bellissima esperienza. A Stefano e Marco per aver condiviso la quotidianità. Alle persone incontrate lungo il cammino e a quelli che mi sono stati vicini nonostante le distanze: Ruben, Sanjay, Giulia, Letizia, Ema, Aida, Elisa, Laura.

Contents

Abstract

The Internet is a global interconnection of networks representing nowadays one of the most important telecommunication technologies. Born as an U.S. military project, it has evolved in a worldwide communication system used by people every day. This success is based on its "freedom" since no single organization or administration entity governs or maintains it. This freedom also motivates the huge heterogeneity of Internet services available today ranging from working activities (e.g., VoIP, e-mail, etc.) to entertainment (e.g., video games, streaming, peer-to-peer, etc.) and commerce (e.g., Amazon, eBay, etc.) just to name a few. The Internet is a fertile and in constant evolution system. Every year new services and software platforms are launched affecting not only the users' activities (e.g. social networks) but also the internal architecture of the networks (e.g., Content Delivery Network vs peer-to-peer) or the devices used to access to the services (e.g., PC vs smartphones and Internet tablets).

The richness of the Internet scenario is paid at the cost of its internal complexity. Eric Schmidt, the CEO of Google, said: *"The Internet is the first thing that humanity has built that humanity doesn't understand, the largest experiment in anarchy that we have ever had."*[1]. At the origins, the Internet has been designed to operate on few standardized services. None could have i) foreseen the success of this media and ii) designed the network to cope with the plethora of nowadays services. If on the one hand this diversity provides the Internet with a certain level of resiliency and has driven innovation, on the other hand understanding its internal mechanisms is a daunting task, made worse by the fast and constant deployment of new services and applications.

However, behind what it could seem a chaotic scenario, the Internet is composed by well defined markets in which big players participate having precise interests:

Users, representing the majority of the people which assess to the network. They are interested in *Quality of Experience* - QoE, i.e., having good performance when accessing to the network, avoiding for example long delay related to the initial buffering when streaming a video. They are also interested in the

[1] `http://www.brainyquote.com/quotes/authors/e/eric_schmidt.html`

Network Neutrality, preserving their freedom to use the Internet independently from which service they are accessing;

Internet Service Providers - ISP, corresponding to organizations which provide Internet access to the customers. They are interested in incrementing the revenues through i) *network engineering* as to optimize the offered services and ii) studying the users' activity as to find new *billing policies*;

Content providers, corresponding to organizations which sell a specific Internet service, e.g., video streaming, file hosting, etc. As for ISPs, they are interested in finding new way to make revenues. At the same time, they have to cope also with illegal activities as *content piracy*, a common flaw since the early days of peer-to-peer systems;

Government regulation agencies, corresponding to organizations which regulate some aspects of the Internet activities. For example, they study *Service Level Agreements* - SLA between users and ISPs, comparing the quality of the Internet access offered to the users with respect to the specifications written in the contract signed.

Other activities as *security* are important for more than one player. Consider for example *malware* and *Denial of Service* - DoS attacks. These can violate the users' privacy, damaging the network and violate some laws. Overall then, there are several motivations to be interested in studying the Internet.

Since the early days, the scientific community has made giant steps toward understanding the Internet. We can generalize that two requirements have to be satisfied. First of all, we need *tools and methodologies* as to inspect and characterize the traffic at different granularities, i.e., per-packet, per-flow, per-port, per-user, etc. In particular, *traffic classification* is one of most important activities performed by network operators. It allows to identify which application has generated a given communication and to study not only the whole network traffic aggregate but also how different applications participate in the composition of the total traffic.

Leveraging on these tools and methodologies, we can further drill into performing *users and network characterization*. For example, monitoring the traffic over long-term periods, we can study the applications' popularity trends and identify the rise of new technologies. We can perform *anomaly detection*, i.e., study unexpected network condition that might be related to either security issues of malfunctioning hardware. We can optimize routing policies, study inter-ISP traffic, investigate the energy consumption of the network elements or work on caching schemes related social network content, just to name a few of the huge amount of research studies recently conducted in the literature.

In this thesis, we present our contributions in studying the Internet discussing the tools and methodologies developed to characterize the network traffic. The thesis is divided in two parts. In the first part we focus on traffic classification methodologies starting from the problem definition and the available solutions in the literature as reported in Chapter 1. In the remaining of the first part we focus on KISS, a novel traffic classification technique we propose based on *Stochastic Packet Inspection* (SPI) analysis. In particular, in Chapter 2 we describe the framework used by the classifier which is then validated in Chapter 3 and 4 for UDP and TCP traffic respectively. Chapter 5 is about the comparison of KISS with other state of the art traffic classifier while in Chapter 6 we extend the KISS framework with some clustering techniques.

Overall, KISS allows to reach a high level of accuracy in traffic classification which is comparable or even better with respect to other traffic classifiers. It presents a flexible structure which is able to identify a rich set of applications with a limited amount of resource requirements.

In the second part of the thesis we study YouTube, the famous video streaming system. Leveraging on Tstat, a passive traffic analyzer, we developed a methodology to identify the YouTube video downloads and we conduct an in depth analysis of many aspects of YouTube. In Chapter 7 we start presenting an overview of the system and its components showing the internal mechanisms adopted. Chapter 8 reports an analysis of the available methodologies in the literature to study YouTube and presents our methodology based on monitoring the real users' activities considering different location, access technologies and devices. In the remaining chapters we present the results of our analysis grouped in four different areas of interest: video content properties (Chapter 9), internal load balancing and caching policies (Chapter 10), users' habits and behaviours (Chapter 11), and download performance (Chapter 12).

Results show that YouTube is a complex system where several components interact with precise policies used to control the communications. Besides its great success, the system is far from being perfect and there is space for further optimization. For example, mobile devices suffer more impairments during the download with respect to PCs. Users stick to the default video resolution and are not interested in changing the quality during the playback. Instead, it is common the abruptly abort of the download. This behaviour is particularly critical because, coupled with aggressive buffering policies used to ensure continuity in the playback, it leads to waste a non negligible amount of traffic, i.e., the users download a portion of the video which it is never played.

Part I

Traffic Classification

Chapter 1

Introduction to Traffic Classification

In this chapter we introduce the reader to the traffic classification. We start defining the problem, underlining the importance of this activity for the modern data networks and the components of a traffic classifier. Then, we present a dissertation on the available solutions in the literature starting from the early beginning to the newest technologies. We conclude defining the metrics used to assess the quality of a classification tool.

1.1 Problem definition

The traffic classification is the process of associating to a given data communication the application that has generated it. Given a traffic aggregate composed by communications generated by a mix of users using different applications, through the traffic classification we group the communications based on some similarity concept.

In a complex scenario as the nowadays Internet, this is a strategic activity especially for Internet Service Providers (ISP) since it allows to obtain important information on both the users' behaviour and the network state. Consider for example the popularity of the Internet services. Knowing how much a service is used is important as to define *billing policies* or to optimize *Quality of Service - QoS* policies. Starting from the traffic classification, we can perform important analysis as *anomaly detection*, identifying unexpected conditions associated to malware or Denial of Service (DDoS) attacks. Similarly, we can operate at *security* level as to identify illegal activities like *content piracy*, particularly critical since the raise of P2P systems.

Through *continuous monitoring*, all the analysis previously mentioned can be extended identifying trends. For example, from the evolution of the application

popularity we can identify the raise or drop of some service. This information might crucial for an operator leading to *network re-engineering*. Consider for example High Definition (HD) video streaming services which require a huge amount of bandwidth both at the last mile and (possibly) in the backbone. A sudden increase in the demand of these services could be reflected in the need of an internal restructure of the operator's network. New commercial partnership might be suggested as well. For example the deploy of local Content Delivery Network (CDN) caches can improve the *Quality of Experience (QoE)* perceived by the users or *Service Level Agreement (SLA)* between different operators. Overall then, the traffic classification is a fundamental step toward the characterization of a data network.

1.1.1 Classification objects and phases

The traffic classification is typically used by network providers or administrators interested in study the traffic that is going to or coming from a set of internal hosts. It requires to access to aggregates of traffic either in real time or with traffic traces. In the first case, we are analysing the actual traffic flowing in the network, while in the second case the traffic is collected in data files for later process. Both the approaches are useful with respect to the type of analysis we are interested in. For example, real time processing is useful for *monitoring* activities and to study the traffic trends, while traffic traces can be used to investigate on specific activities.

As to classify the traffic, we need to define which are the classification objects. A traffic classifier can elaborates three different type of objects:

- **flow**, corresponding to a directional communication in which an host A sends traffic to a specific host B. Formally, this means all the packets sharing the tuple (IP_A, IP_B, $L4_{portA}$, $L4_{portB}$);
- **connection**, corresponding to a non directional communication, in which we are grouping two flows having opposite directions. Formally, we are grouping the flows with either (IP_A, IP_B, $L4_{portA}$, $L4_{portB}$) or (IP_B, IP_A, $L4_{portB}$, $L4_{portA}$);
- **endpoint**, corresponding to all the flows and connections generated by a single host. Formally this means all the communications sharing the tuple (IP, $L4_{port}$).

The three objects allow to inspect the traffic at different levels of granularity and have to be used depending on the type of applications we are interested in. For example, one can be interested in identifying a single flow (as in the case of a VoIP stream), or in detecting an endpoint and therefore all packets sent/received from it (as in the case of a P2P application).

Once the classification object is defined, then three phases are required as to classify the traffic:

- **Feature extraction**, the process of characterizing the traffic. In this phase, we extract a subset of information called *features* which characterize the communications of a given application;
- **Decision process**, the process of modeling and defining the application label. In this phase, we exploit algorithms elaborating the traffic features and creating a mathematical model. The classification then is performed checking if a data communication match the generated model;
- **Accuracy evaluation**, the process of verifying the goodness of the classification performed.

The feature extraction and the decision process correspond to the core of each classifier. These two phases are orthogonal and in the literature there is a wide range of approaches which can be used, as we will see in Sec. 1.2. As to evaluate the classification accuracy, some standard metrics are available as detailed in Sec. 1.3.

1.2 State of the art

As to classify the traffic, a proper characterization of each application's protocol is needed. This requires a deep *domain knowledge* of the internal mechanisms exploited by each applications. Unfortunately, it is difficult to acquire such knowledge since the majority of the applications are not "open" i.e., they do not expose neither the protocol specifications nor the other mechanisms exploited in the communications. This scenario is further exacerbated by the adoption of *encryption* and *tunneling* mechanisms as to protect the users' privacy and increase the complexity of the classification task. Moreover, the classifiers need to cope with the increase of the traffic volume and higher bandwidth rates. Overall then, traffic classification is a daunting task: it requires lightweight, scalable and accurate algorithms which do not expose sensible information of the users' activity.

Given this complexity, traffic classification is a hot topic for the networking research community. A rich set of methodologies has been developed in the last decade [5, 6, 16, 17, 21, 25, 26, 32, 38, 54, 40, 48, 51, 55, 53, 59, 64, 69, 73, 53, 37, 39] but the "perfect oracle" is still far from being developed. In the following, we present a dissertation on the most important methodologies, comparing keys aspects of different solutions. We aim not to a complete survey on the subject but rather to introduce the reader to the several aspects related to the traffic classification. To better structure this overview, we divide the methodologies in a few categories according to the information on which the classification is based.

Port-based classification: in the early days, only few services were available on the Internet and most of the them run on a specific transport-layer port.

In such a scenario, *port-based* classification was the most straightforward approach: each TCP/UDP port is mapped to a specific application (e.g., TCP:80 = HTTP, UDP:53 = DNS). Unfortunately, in the nowadays Internet such mechanism has become unreliable [36, 53] due to i) the proliferation of applications using random ports (e.g., P2P or VoIP); ii) tunneling mechanisms based HTTP and adopted by the applications as to easily traverse middleboxes such as NAT or firewall. Overall, port-based mechanisms fail because of their too naive mapping strategy which do not capture the specific nature of each application.

Deep Packet Inspection - DPI: Deep Packet Inspection (DPI) tools exploit *pattern matching* and *regular expression* looking for keywords inside the packet payload. For example, if the payload starts with the string `'BitTorrent'` than it is very likely that the flow has been generated by the BitTorrent application. In this case, the feature extraction process is represented by the definition of the matching rules which are collected in a dictionary. Instead, the decision process is based on checking the list of rules until a match is found.

This mechanism can be extended with more complex rules that require the analysis of a sequence of packets correlating also the two directions of communication. For example, considering the HTTP protocol, the `GET` command sent by a client is expected to be followed by a reply (e.g., `200 OK`) sent by the server contacted. In this case the classification mechanism corresponds to a *Finite State Automaton - FSA* that reflects the internal states of the application's protocol.

DPI tools represent the most used technology today with several solutions either commercial [56, 65, 57, 18] or Open Source [72, 58, 43]. However, despite this popularity they present important limitations:

- the identification of the classification rules requires a complex *reverse engineering* of the application mechanisms and the robustness of the rules can be weak. In fact, application's update can introduce modifications at protocol level or new features can be added compromising the identification rules and the classification accuracy. It follows that DPI tools require a continuous maintenance;
- packet inspection is not able to cope with protocol encryption or obfuscation since in this case the payload corresponds to a random sequence of number;
- payload inspection techniques are hardly scalable. In fact, they require too much memory and computation resources to be deployed on backbone links;

- the packet payload usually contains sensible information about the users' activity. Given the possible privacy violation, this methodology rise some questions about the validity of the approach by the Internet users.

Statistical classifiers: this class of methodologies leverage on a statistical characterization of the traffic of each application without the need of payload inspection or complex reverse engineering operations. The rationale behind this approach is that different Internet services present different traffic patterns. Compare for example generic web traffic and a VoIP call. In the first case the traffic is composed by shorts and irregular communications where the majority of the volume is downloaded from the Web server. A VoIP call instead presents packets of specific size with a proper interleaving, while the volume is usually symmetrical.

A statistical classifier is based on the analysis of *protocol feature* i.e., a statistical representation of both the protocol characteristics and the dynamic of the application's communications. This characterization is then further processed using different techniques

- **Unsupervised Machine-Learning**: Unsupervised Machine-Learning algorithms, from which clustering and neural networks are the most famous, are data mining techniques used very often in the field of traffic classification [4, 51, 6, 19, 22]. They allow to group communications looking for a hidden structure of unlabeled data. In this case, the decision process in based on a *training* which operates without any knowledge of the mix of applications to classify. For example, given an aggregate of traffic obtained from 5 different applications, in the best case an algorithm is able to split the communications in 5 groups. The quality of the classification is related to the *homogeneity* of each group i.e., the more it contains only points of a single application, the better the accuracy.

  Even if these algorithms can easily group the communications, they cannot identify which application is related to each group. As such, to evaluate the accuracy of the classification we need a *ground truth*. Moreover, tuning the parameters can be tricky since it is difficult to map them with respect to the expected output;
- **Supervised Machine-Learning**: in this class we can find techniques as Naive Bayes [54] classifiers, C4.5 and other Decision Trees algorithms [41] or Support Vector Machines [5, 25]. As for the previous class, also this family of algorithms is borrowed from the data mining research field. Differently from unsupervised techniques, in this case the training process requires that the flows are pre-labeled i.e., the flows have to be already divided is classes. This pre-process allows to gain an higher classification

		Oracle Classification	
		True	False
Classification Result	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Table 1.1. Definition of False/True Positive and False/True Negative

accuracy but it also represents the major drawback of these techniques requiring tedious manual inspection of the traffic or to rely on DPI tools as to have a ground truth.

- **Behavioural classifiers**: behavioural classification techniques [38, 37, 73] target the classification of Internet hosts on the sole basis of the transport layer traffic patterns they generate. For example, P2P hosts contacts many different hosts typically using a single port, whereas a Web server is contacted by different clients with multiple parallel connections. This description can be obtained with different level of granularity, from the simple count of the distinct ports used by each application [73, 37] to more complex solutions in which graphs are used to track the connections between endpoints [33, 35].

Compared to DPI tools, these class of algorithms are less "intrusive", more automated, and require less resources. Anyway, they usually have a lower accuracy which justify the very slow adoption process of these technologies.

1.3 Testing methodology and metrics

Once a classifier has been designed, its performance must be evaluated and proper metrics must be defined. Assessing the performance of Internet traffic classifiers is not a trivial task due to the difficulty in knowing the *ground truth*, i.e., what was the actual application that generated the traffic [53]: for the ground truth, an *oracle* is needed. Testing the classification engine by means of artificial traffic (e.g., by generating traffic in a testbed) solves the problem of knowing the ground truth (you are the oracle), but reduces the representativeness of the experiments, since synthetic traces are hardly representative of real world traffic. Assessing the performance against traffic traces collected from operative networks is therefore mandatory. To extract the ground truth from real traces we can exploit ad-hoc classifiers (usually DPI) or inspecting manually the traffic connections. However, the oracle may still be fooled.

To quantify the accuracy of the classification, we can measure False Positive (FP) and True Positive (TP), and the False Negative (FN) and True Negative (TN). A

test is said "True" if the classification result and the oracle are in agreement. A test is said "False" on the contrary. The result of a test is "Positive" if the classifier accepts the sample as belonging to the specific class. On the contrary a test is "Negative". For example, consider a flow. The oracle states that this flow is an eMule flow. If the flow is classified as an eMule flow, then we have a True Positive. If not, then we have a False Negative. Consider instead a flow which is not an eMule flow according to the oracle. If the flow is classified as an eMule flow, then we have a False Positive. If not, then we have a True Negative. Table 1.1 summarizes the definitions.

The corresponding percentages must be evaluated as

- **False Positive** Percentage (%FP) is the percentage of *negative* samples that were erroneously reported as being positive:

$$\%FP = 100 \cdot \frac{FP}{Total\ Number\ of\ Negative\ Samples} \tag{1.1}$$

- **False Negative** percentage (%FN) the proportion of *positive* samples that were erroneously reported as negative:

$$\%FN = 100 \cdot \frac{FN}{Total\ Number\ of\ Positive\ Samples} \tag{1.2}$$

- **True Positive** Percentage (%TP) is 100 - %FN;
- **True Negative** Percentage (%TN) is 100 - %FP;

Other two important quality indexes are usually considered:

- **Recall**, representing the completeness of the classification, an defined as %TP;
- **Precision**, corresponding to exactness or fidelity of the classification and defined as

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{1.3}$$

These two measures complement each other. In fact, a precision of 1.0 for a class C means that every item labeled as belonging to C does indeed belong to C. It however says nothing about the number of items from class C that were not labeled correctly. A Recall of 1.0 means that every item from class C was labeled as belonging to class C. It however says nothing about how many other items were incorrectly labeled as belonging to class C.

Results are often expressed by means of a *Confusion Matrix*. In the field of artificial intelligence, a confusion matrix is a visualization tool typically used in

		Classifier		
		Emule	Bittorrent	*Recall*
Oracle	Emule	90	10	90/(90+10)=0.9
	Bittorrent	50	50	50/(50+50)= 0.5
	Precision	90/(90+50) = 0.64	0/(0+10) = 0	

Table 1.2. Example of confusion matrix.

machine-learning. Each column of the matrix represents the instances in a predicted class as stated by the oracle, while each row represents the instances in an actual class. One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e., commonly mislabeling one as another). In fact, in the best case all the samples are correctly classified, the matrix presents values only on the main diagonal. Consider for example Table 1.2 where a simple confusion matrix is reported for the classification of 100 flows of Emule and BitTorrent. We can observe that the classifier present some errors since there are values outside the main diagonal. In particular, while for Emule traffic the classification is very accurate (Recall=90%), this is not true for BitTorrent traffic (Precision=0.64).

The example reported allows to highlight an important difference between Recall and Precision. While Recall is applied to each specific class of traffic measuring the homogeneity of the classification, Precision measures how different types of traffic are classified with respect to a specific class. In other words, since Recall is applied to the rows of the table, the number of samples in each class can be different. This instead cannot be true if we want to measure the Precision, otherwise the results are biased.

Chapter 2

KISS: Stochastic Packet Inspection

2.1 Introduction

In the previous chapter we introduced the traffic classification. We have seen that, despite the effort devoted to this activity and the significant progress in the field [69, 53, 48, 64, 54, 10, 6, 16, 38, 73, 5], the ultimate and definitive solution is still far from being available. DPI tools are still regarded as the state of the art and deployed in practice, even if it is well known that the proliferation of proprietary and evolving protocols and the adoption of strong encryption techniques are deemed to make DPI ineffective. Conversely, alternative solutions based on a statistical characterization of the traffic solve some of the issues of DPI classifiers but are usually less precise or target a limited set of applications.

In this chapter we focus on *Stochastic Packet Inspection* (SPI), a new methodology which try to fuse the benefits of the payload inspection used by the DPI with the flexibility of a statistical classifier. The intuition behind SPI is that an application-layer protocol can be identified by statistically characterizing the values observed in a stream of packets. Consider for example the problem of identifying foreign languages. One may try to differentiate the languages based on a dictionary of known words (e.g., "Thanks" for English, "Merci" for French, "Grazie" for Italian and so on). This is the typical approach used by DPI where we inspect the packet payload looking for keywords related to different application-layer protocols. A different approach instead can be based on the analysis of the cacophony of the conversation e.g., observing the frequencies and the position of symbols like "x", or "h", or "i" in the communication. In this case we exploit *syntax* properties of the language while ignoring the *semantic*.

The application of SPI has been already proved successful in assisting the identification of particularly tricky traffic such as Skype [10]. In this work, we push this intuition further generalizing the methodology to the identification of any application running on both UDP and TCP. This is achieved by statistically characterizing the frequencies of observed values in the packet payload, by performing a test similar to the Pearson's χ^2 test. The results of the test are then used to compactly represent application fingerprints, which we call Chi-Square Signatures - ChiSS (pronounced as KISS).

While KISS fingerprints stem from packet inspection, they have several advantages over classical DPI signatures:

- they can be automatically derived, i.e., no cumbersome and tedious reverse engineering is required;
- they can be quickly updated, so that they are well apt to the context of fast-evolving Internet applications;
- they are easily portable across different network settings, since fingerprints depend solely on the L7-protocol format;
- they are robust to routing asymmetry, packet loss or sampling, retransmission, or any possible strange packet arrival pattern, since they build over a statistical characterization of protocol format rather than on a deterministic description;
- they are suitable to both per-flow and per-endpoint classification;
- their computational and memory requirements are limited, so that they are suitable for on-line classification.

However, KISS shares with DPI classifiers the need to look at application layer messages. As a drawback, in case of encrypted payload, both approaches become ineffective.

After fingerprints have been extracted, proper classification must be achieved, i.e., individual items should be placed into the most likely class. A huge set of methodologies are available from the literature, from simple threshold based heuristics [38], to Naive Bayesian classifiers [10, 54], to advanced statistical classification techniques [15]. In this work, we compare a simple geometric decision process based on Euclidean distance with Support Vector Machines (SVMs) [15], which are well known in the statistical classification field, but have been rarely exploited in the context of Internet traffic classification.

In the remaining of the chapter, we detail the KISS framework. We start presenting the internal workflow of the classifier, showing how the classification is achieved starting from some traffic. Then we focus on the feature extraction process and

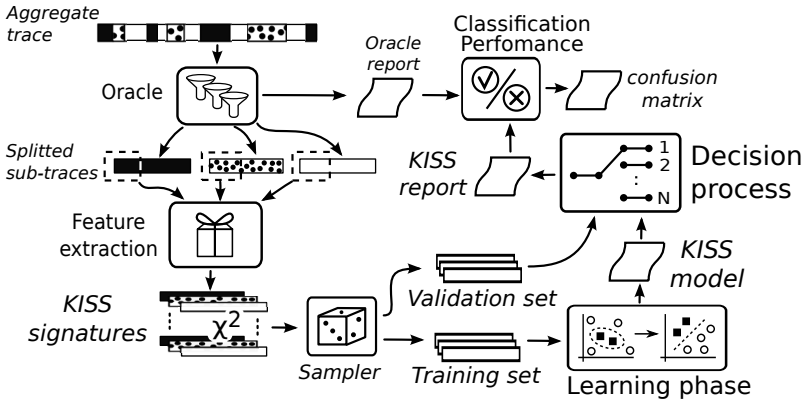


Figure 2.1. Schematical representation of the KISS workflow.

the decision process, detailing the mathematical formulation and the methodologies exploited as to describe the Internet traffic.

2.2 KISS workflow

The components of the KISS workflow are reported in Figure 2.1. As we can see, the internal structure of the classifier is complex and some steps are required as to obtain the classification.

Oracle: starting from an aggregate traffic trace, the Oracle split the traffic is sub traces, one for each application we are interested in the classification. This process can be done manually, but it more common to rely on an ad-hoc DPI. Beside the sub traces, when the splitting process is concluded, the Oracle generates a report specifying the application associated to each communication. This report corresponds to the *ground truth* that will be used to assess the classification performance;

Feature extraction: a statistical analysis is applied to the streams of packets contained in the sub-traces obtained from the Oracle. This process allows to extract the *features* that characterize each application which are collected in the KISS signature;

Learning phase and decision process: a subset of the signature are used as *training* set for a supervised machine-learning algorithm that generate the KISS classification model. This model is then used to classify the remaining portion of the signatures obtaining then a classification report;

Classification performance: the classification report obtained from the Oracle is compared with the report obtained from KISS and a confusion matrix is generated detailing the classification performance.

2.3 Feature extraction

A traditional DPI classifier inspects packet payload looking for deterministic patterns, such as particular strings which are compared to those in a signature database. The process of defining the signatures is a complex task that requires a deep knowledge of the protocols that need to be identified. As such, any changes in a protocol can invalidate the signature, which becomes outdated and must manually redefined.

The goal of KISS is instead to automatically discover application-layer header format, without caring about specific values of the header fields: we aim at automatically let the protocol format emerge. In fact, considering a protocol header, we can identify a specific layout of fields such as constant identifiers, counters, words from a small dictionary (message/protocol type, flags, etc), or truly random values coming from encryption or compression algorithms. These coarse classes of fields can be easily distinguished through a simple statistical characterization of the values observed in a sequence of packets (identifiers are constant, counters increments according to step, etc.).

Let's suppose that each packet payload starts with an application protocol header. Focusing on the first bytes of the payload, we can create statistical fingerprint of the protocol header with a number of different metrics, such as the Entropy measure, the Pearson's χ^2 measure, the Kullback-Leibner divergence, etc. In particular KISS is based on the application of a simple Chi-Square like test. The test originally estimates the goodness-of-fit between observed samples of a random variable and a given theoretical distribution. Assume that the possible outcomes of an experiment are K different values; and O_k are the empirical frequencies of the observed values, out of C total observations ($\sum_k O_k = C$). Let E_k be the number of expected observations of k for the theoretical distribution $E_k = C \cdot p_k$ with p_k the probability of value k. Given that C is large, the distribution of the random variable

$$X = \sum_{k=1}^{K} \frac{(O_k - E_k)^2}{E_k} \tag{2.1}$$

that represents the distance between the observed empirical and theoretical distributions, can be approximated by a Chi-Square, or χ^2, distribution with $K-1$ degrees of freedom. In the classical goodness of fit test, the values of X are compared with the typical values of a Chi-Square distributed random variable: the frequent occurrence of low probability values is interpreted as an indication of a bad fitting. In KISS, we build a similar experiment analyzing the content of groups of bits, called

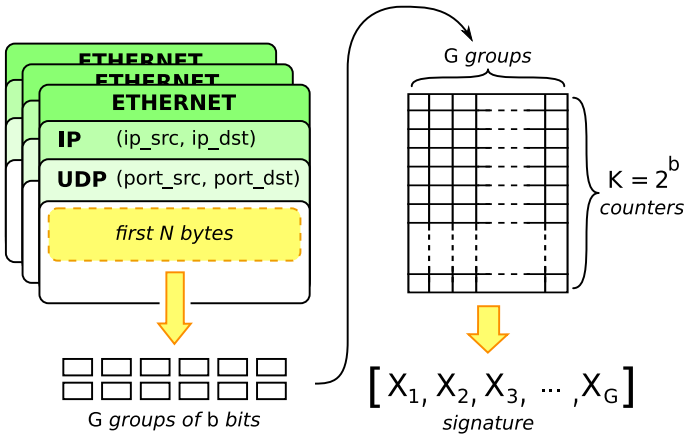


Figure 2.2. Scheme of signature extraction process for UDP traffic

chunks, taken from the packet payload we want to classify. Intuitively, if the group of bits corresponds to a constant value, than the distribution is far from being uniform. Conversely if only some bits are constant, then the distribution is closer to be uniform.

As to perform such analysis, we need to identify the application protocol header. inside the packet payload. This process is strictly related to the transport protocol adopted by the application. In fact, since UDP is a connectionless protocol, the first bytes of the payload of each UDP packet typically contain application layer protocol header. Conversely, in case of TCP, the segmentation introduced at the transport layer reduces the chances that a TCP segment carries the application protocol header. However, we can expect that the very few segments after the TCP handshake carries information that are specific to each protocol (e.g., as in behavioral classification approaches [6, 16] that exploit the size and arrival time of the first few packets of a flow).

2.3.1 KISS Signature for UDP Traffic

Figure 2.2 reports the scheme of the process used to generate the KISS signatures for UDP traffic. We start considering a *stream* of packets corresponding to either a specific flow or endpoint (see Chapter 1.1.1). The first N bytes of each packet payload are divided into G *groups* of b consecutive bits. Each group g corresponds to a *chunk* and can take integer values in $[0,2^b - 1]$. From packets of the same stream, we collect, for each group g, the number of observations of each value $i \in [0,2^b - 1]$

denoted as $O_i^{(\mathrm{g})}$. We then define a window of C packets, in which we compute

$$X_g = \sum_{i=0}^{2^b-1} \frac{\left(O_i^{(\mathrm{g})} - E_i^{(g)}\right)^2}{E_i^{(g)}} \tag{2.2}$$

and collect them in the vector

$$\overline{X} = [\, X_1, X_2, \cdots, X_G \,] \tag{2.3}$$

which is the KISS signature.

Suppose for example, to consider chunks of 4 bits ($b = 4$) and to consider the first chunk of the payload. For a window of C packets, we are extracting the first chunk and counting how many times it assumes value 0,1,...,15. In the end, Eq.(2.2) measure the distance between the empirical distribution of the observed values with the expected one. One possibility to characterize a given protocol is to estimate the expected distribution $\{E_i^{(g)}\}$ for each group g, so that the set of signatures are created by describing the expected distribution of the protocols of interest. During the classification process then, the observed group g distribution $\{O_i^{(g)}\}$ must be compared to each of the $\{E_i^{(g)}\}$ in the database, for example using the Chi-square test to select the most likely distribution. However, this process ends up to be complex since there should be too many distribution to compare. Moreover, it presents the same rigidity of a DPI classifier. Suppose for example that the protocol has a "flow ID" in group 1. Consider now two flows, one used for training and one for testing, generated by the same application. Let the training flow take the value 12 in group 1 while the test flow take instead the value 1 in the same group. Clearly, the comparison of the two observed distributions does not pass the Chi-square test, and the test flow is not correctly classified as using the same protocol as the training flow.

For the above motivations, we propose to simply check the distance between the observed values and a reference distribution, which we choose as the uniform distribution, i.e., $E_i^{(g)} = E = \frac{C}{2^b}$. In the previous example, the group randomness of the two flows has the same X_1 values, that identify a "constant" value, independently of the actual value. In other terms, we use a Chi-Square like test to measure the randomness of groups of bits or as an implicit estimate of the source entropy.

To give an intuition of how (2.2) evolves versus C, consider the case in which a deterministic group of bits is observed. Since for a deterministic group only one value is possible, the value of X_g becomes

$$X_g \;=\; \sum_{i=0}^{2^b-1} \frac{\left(O_i^{(\mathrm{g})} - E_i^{(g)}\right)^2}{E_i^{(g)}} \tag{2.4}$$

$$= \sum_{i=0}^{2^b-1} \frac{(O_i^{(g)} - E)^2}{E} \tag{2.5}$$

$$= \frac{(C-E)^2 + \left(2^b - 1\right) E^2}{E} \tag{2.6}$$

$$= C\left(2^b - 1\right) \tag{2.7}$$

Then, X_g linearly increases with C.

In general, for a block in which b_0 bits are constant, it can be shown that

$$X_g = C\left(2^{b_0} - 1\right) + 2^{b_0}\chi_A^2 \tag{2.8}$$

where χ_A^2 is the Chi-Square with A degrees of freedom. In this case, $A = 2^{b-b_0} - 1$.

To provide an example of the evolution of X_g, left plot in Figure 2.3 reports the value of two 4-bit long chunks belonging to two streams of two different traffic protocols, namely DNS and eMule, versus the number of collected packets C. The steep lines corresponding to groups taken from the eMule stream refer to fields that are almost constants. In this case, the longer the experiment is (larger C), the larger the distance from the uniform distribution is, i.e., the bits are far from being uniformly distributed. In the same plot, observe the lines referring to DNS traffic. The lowest one has a very slow increase with C, its behavior is almost perfectly random, the values of X_3 being compatible with those of a Chi-Square distribution. The bouncing line, instead, corresponds to the typical behavior of a counter. The computation of Eq. (2.2) over consecutive bits of a counter cyclically varies from very low values (when all the values have been seen the same number of times) to large values. The periodicity of this behavior depends on the group position inside the counter, while increasing the number of packets observed the distribution decays because the differences between the observed frequencies are less important.

While randomness provides a coarse classification over individual groups, by jointly considering a set of G groups through the vector $\overline{X}$ the fingerprint becomes extremely accurate. Observe right plot in Figure 2.3. Each point in the figure corresponds to a different stream. A window of $C = 80$ packets is used to derive the signatures using the couple of features (X_2,X_3) as coordinates. Points obtained from DNS streams are displayed in the low left corner of the plot; points from eMule are spread in the top part of the plot. Notice also that signatures of the same protocol class are not identical. This is due to both the behavior of each application, and to different implementations of the same protocol. For example, some eMule clients can be downloading, uploading, or waiting, therefore exchanging different types of messages. Similarly, different implementations of a DNS server can use different random number generators to extract the query identifier. It is the scope of the decision process to define the areas where points of the same protocols are expected.

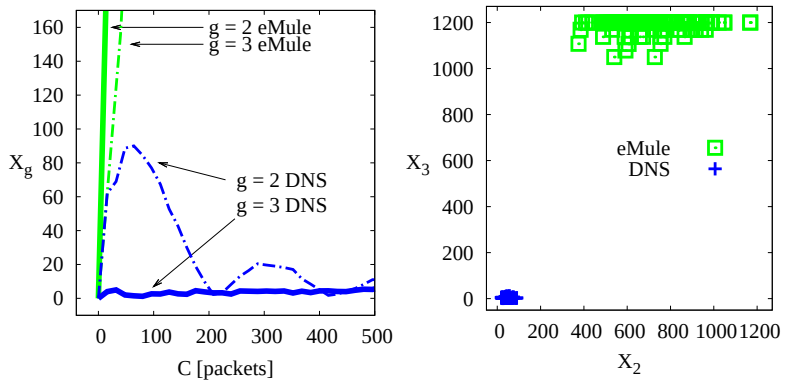


Figure 2.3. Evolution in time (left) and dispersions in space (right) of X of two groups of 4 bits extracted from the second byte of UDP payloads.

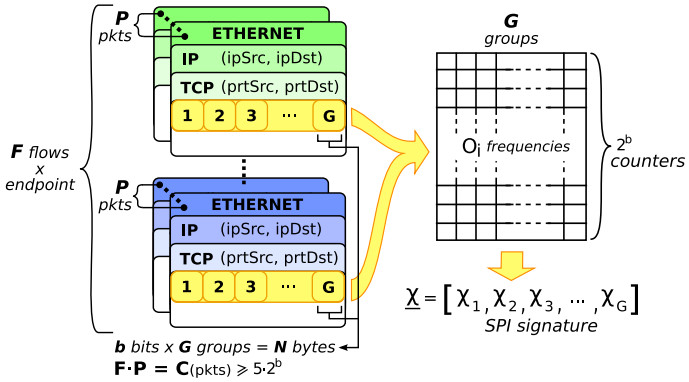
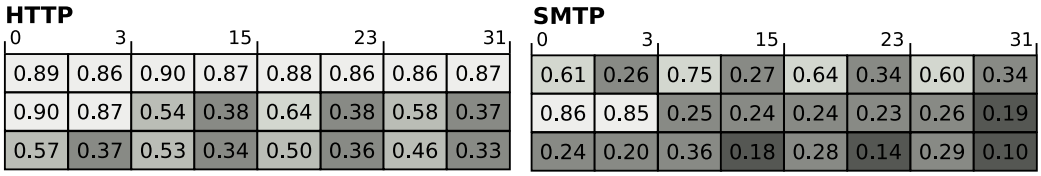


Figure 2.4. SPI signatures of TCP traffic: G groups of b-bits long chunks are extracted from the first P packets of F different flows (with $C = F \cdot P$) originated from (or destined) to the same endpoint (IP, TCP_{port}).

Intuitively, different protocols fall in different areas that are clearly identified and easily separable: a simple straight horizontal line can effectively separate the two regions considering this example. In general, more complex surfaces have to be found.

2.3.2 KISS Signature for TCP Traffic

Differently from UDP, when considering TCP traffic only endpoints can be used as classification objects. In fact, even if the application deliver packets with an application header, because of the fragmentation operated by TCP it very complex to retrieve such information. We can exploits only the first few packets after the end of



(a) (b)

Figure 2.5. Example of SPI signatures of HTTP (a) and SMTP (b) protocols, server endpoint is the destination.

the handshake which are very likely to carry an application header. This means that a single flow is not enough to compute a KISS signature and we need to aggregate different flows generated from or destined to the same endpoint (IP, TCP_{port}). We assume to be at the edge of the network, where all the endpoint traffic transits, and separately consider the two traffic directions, i.e., the traffic directed to, and the traffic originated from the endpoint (IP, TCP_{port}). As reported in 2.4, signatures are computed by observing the G groups of chunks over the first P packets of F consecutive flows originated from (or destined to) the same endpoint, where P and F satisfies $C = P \cdot F$.

An example of KISS signatures for two different protocols, namely HTTP and SMTP is given in 2.5. The classical header representation is adopted, representing chunks in network order from left to right, top to bottom. Four bytes are reported on each row (i.e., 8 chunks) and, for reference, bit offsets are reported at the top. Each chunk reports the X_g value normalized, which is also visually represented in a gray scale. Lighter colors correspond to higher values of X_g, suggesting deterministic fields, while darker colors correspond to low values of X_g, hinting to random fields. Comparing 2.5(a) and 2.5(b) we can confirm that, though the randomness test provides only a coarse classification over individual groups, and expressive fingerprints can be built by considering the whole set of G chunks. This allows to clearly differentiate between protocols.

To grasp the KISS signatures expressiveness, let us first consider the case of the Web service, implemented over the simple and stateless HTTP protocol. In this case, requests directed toward the server often begins with "`GET /`": the high occurrence of this 5-characters string translates into the first 10 chunks to be almost deterministic (high X_g values). Variability of the first chunks is due to the fact that server can receive other HTTP methods than `GET` (e.g., `POST`, `HEAD`, `PUT`). Variability of subsequent chunks is instead tied to the different resources that can be specified after the method (e.g., URL in case of `GET`, parameters in case of `POST`, etc.).

Interestingly, HTTP uses an ASCII alphabet, which translates also into a reduced set of values which each chunk can take. Given a byte, since we use $b = 4$ bits long

chunks, an ASCII encoded character is split into two chunks, corresponding to the most and least significant part of the byte respectively. The most significant chunk shows higher determinism ($X_g \simeq 0.6$), while least significant chunk shows higher randomness ($X_g \simeq 0.3$). For example, consider the ASCII uppercase letters {A,...,Z} which take hexadecimal values in $\{0 \times 41, \ldots, 0 \times 54\}$. The most significant bits of a character fall into a chunk that takes only values of 4 and 5. Conversely, least significant bits falls into a chunk that takes any possible values from 0 to 15. This leads to different X_g values, i.e., a different randomness. In Figure 2.5(a), the impact of ASCII encoding can be appreciated by observing the alternation of lighter and darker chunks.

Let us now consider the SMTP protocol signature reported in Figure 2.5(b). Recall that an SMTP client contacts a server with the typical sequence of commands `EHLO`, `MAIL`, `RCPT`, `DATA`. Notice that these commands are 4-characters long (which correspond to 8-chunks) and, with the exception of the `DATA` command, are followed by a space character and some parameters of variable length. Since several commands are used during the same session, there is a larger number of observed symbols, which therefore decrease X_g of corresponding chunks. Also in the SMTP protocol case, commands are encoded using ASCII alphabet, causing a higher X_g value for most significant chunks than for least significant chunks.[1] The highly probable space character at the 5th byte causes the 9th and 10th chunks to take deterministic values, as the high X_g value observed in such position shows. Chunks corresponding to characters after the 5th position may contain any symbol of the ASCII alphabet, (e.g., angle brackets to enclose mail addresses, etc.) or user data, which then decrease the X_g values of corresponding chunks.

2.4 Decision Process

When the feature extraction is completed, the KISS signatures have to be used to i) create a classification model using a *training phase*; ii) classify the data communication using the model created. As suggested by the examples reported in Figure 2.3 and Figure 2.5, KISS signatures are better suited for a geometrical analysis. Considering the X_g values as geometrical coordinates, each signature corresponds to a point in an hyper space. If two applications are different and the signature generated are enough representative, they are associated to clouds of points in two different areas of the hyper space. The classification model then should describe the geometry of the clouds while the classification is straightforward: if a point fall in the area of an application, then it is very likely that it is associated to that application. Instead, if the point fall outside all the areas, then the flows it either forced to one

[1] The higher variability of the first 8 chunks is also due to other possible commands (e.g., `VRFY`), the presence of old clients (e.g., `HELO` instead of `EHLO`), clients using lower case letters, etc.

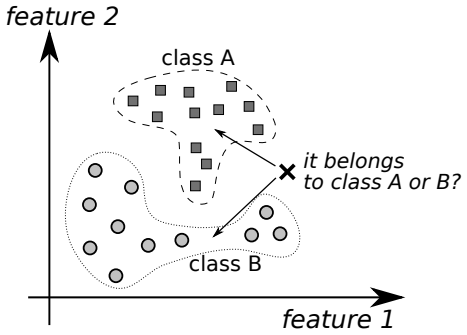


Figure 2.6. Example of geometrical decision process.

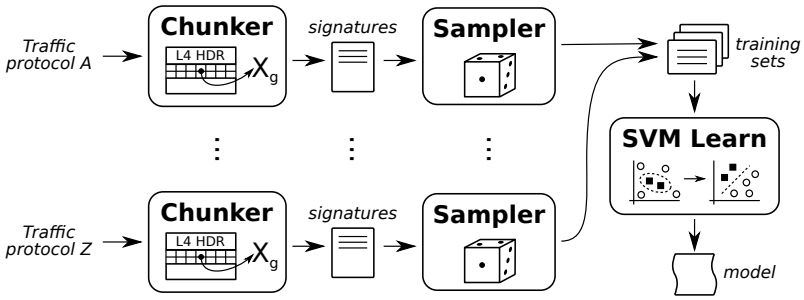


Figure 2.7. Schematic representation of KISS learning steps.

of the known classes (e.g., considering the closest cloud) or it left unlabeled (e.g., it "noise" or correspond to unknown traffic). An schematic example of this process is reported in Figure 2.6.

2.4.1 SVM Decision Process

Among the different machine-learning approaches available (see Chapter 1.2), KISS is based on Support Vector Machine (SVM), a set of supervised learning methods used for classification and regression. The key idea of SVM is to displace the training samples (by means of a transformation from the original N-dimensional space to a possibly infinite-dimensional space) so that samples belonging to different classes can be separated by the simplest surface, i.e., an hyper-plane. This makes SVMs very powerful:

- they are robust to the training set size and composition;
- their computational and memory requirements are very limited during the

classification phase, even if the training phase can be computationally expensive;

- they exhibit a very high discriminating power, so that they typically achieve very high classification accuracy;
- there is a large number of efficient algorithms and implementations already available. In particular, in this paper we adopted the `LIBSVM` [47] implementation.

The SVM learning process is used as reported in Figure 2.7. We start by considering some streams that belong to the set of applications we want to model. Streams are then fed into a *chunker*, whose role is to derive the KISS signatures. Then, the signature sets are randomly sampled by the *sampler*, so as to select the *training set*. The training set is then fed to the learning system after which the KISS model is produced. Notice that the output of the SVM training phase is a definition of a number of regions equal to the number of classes defined during the training phase, e.g., one for each protocol that is offered during the training phase. This implies that a sample will then always be classified as belonging to one of the known classes. Considering traffic classification, an additional region is needed to classify all samples that do not belong to any of the given protocols, i.e., to represent *unknown* protocols. Thus, the training set must contain two types of signatures: i) the ones referring to traffic generated by the applications to classify; ii) the ones representing all the remaining traffic, which we refer to as *background*. It represents the set of applications that we cannot classify or we are not interested in classifying.

Chapter 3

Evaluating KISS on UDP Traffic

We aim at assessing KISS performance in the most difficult scenario, whenever possible. For most of the results we show, we consider real traffic traces, collected from an operative, totally uncontrolled network. In addition, to evaluate the performance of KISS when dealing with new protocols, we also selected, as a case study, P2P-TV applications. Indeed P2P-TV systems have been recently introduced and they are starting to became popular. They rely on proprietary design and protocols, they preferentially use UDP as transport protocol, and they are expected to offer a large amount of traffic to the network; thus, their classification is going to be the more and more important.

3.1 Testing Data sets

Table 3.1 describes the traces used to test the classifier reporting the total amount of bytes, packets, flows and endpoints, and the collection time and duration of each trace.

We assume packets belonging to the same flow/endpoint are exposed to the KISS engine, so that after digesting C packets, a classification decision is taken, and a new observation window begins. Therefore, several classification decisions are possibly taken about a single flow or endpoint. In this paper, we consider independent classifications, so that the same flow/endpoint can be classified differently at each window. Notice that some reconciliation algorithm can be easily designed to increase the accuracy of the classification by considering the set of classifications involving the same flow or endpoint, e.g., adopting a majority criterion.

Notice that considering UDP traffic i) no assumption about observing the first set of packets is stated; ii) there is no need to observe bidirectional streams of packets; and iii) not all packets belonging to the same flow/endpoint must be exposed to the classifier; possible packet drops, reordering, sampling can be present.

	Bytes	Packets	Flows	Endpnts	Time period
RealTrace I	53.13G	321M	18.25M	1.72M	22h, May '06
RealTrace II	31.33G	133M	5.25M	1.02M	12h, Jun '07
NapaTraces	10.25M	14M	132K	48.5K	3h, Apr '08
SkypeTraces	3.7G	24.7M	966	559	96h, May '06

Table 3.1. Description of the UDP data traces

Real Traffic Traces (RealTrace): these traces were collected from the network of an Italian ISP, offering converged services, in which data, native VoIP [9], and IPTV services share a single broadband connection. This network is a very heterogeneous scenario, in which users are free to use the network without any restriction. It therefore represents a very challenging scenario for traffic classification. A probe node based on high-end PC running Linux has been installed in a PoP located in Turin, in which more than 500 users are connected, using more than 2000 different IP addresses (e.g., VoIP phones, set-top-boxes, PCs, etc.). All packets entering/leaving the PoP have been captured. The measurements presented in this paper refer to two data sets that we call RealTrace-I and RealTrace-II, collected in 2006 and 2007.

Both traces contain many popular applications generating UDP traffic, in particular we selected: i) Emule, ii) VoIP (over RTP), and iii) DNS protocols. Indeed, these three protocols account for more than 80% of UDP endpoints, corresponding to 95% of the flows, and to more than 96% of the total UDP volume. In the remaining traffic, nearly 2% of flows are related to BitTorrent accounting for less than 1% of bytes. Skype communications instead present the typical mice/elephant behavior since a negligible number of flows account for more than 1% of the total volume in both traces. Being dated back to 2006 and 2007, no P2P-TV traffic is present;

P2P-TV Traces (NapaTraces): to assess the performance of KISS with P2P-TV traffic, we selected, among the available P2P-TV applications, PPLive, Joost, SopCast and TVants. Since none of the selected applications were available at the time of real traffic trace collection, we are forced to rely on ad-hoc traces. We used a set of traces collected in the context of Napa-Wine [46] project, in which a large scale experiment was organized to observe the performance of the above mentioned P2P-TV applications. We refer to this data set as NapaTraces and consists of packet level traces collected from more than 45 PCs running P2P-TV applications in 5 different Countries, at 11 different institutions. The data set includes traces collected from PCs in Campus LANs, Corporate networks with restrictive policies, home ADSL connections, so that

both nodes with public and private IP addressing are present. We are therefore confident that the heterogeneity of this data set is representative of a wide range of different scenarios;

Skype Traces (SkypeTraces): we also use a public available data set for the Skype traffic [71]. The data set contains both Skype traffic identified in [10] and traces collected in a controlled environment using PCs running different versions of Skype and different operating systems such as Windows, Linux and Pocket-PC.

3.2 The oracle definition

To obtain the ground truth from an *aggregated* trace we developed a DPI classifier that was explicitly designed. It was implemented in Tstat [24, 72], an its performance were manually fine tuned and double checked. In particular, DPI rules can be summarized as follow:

- DNS: we rely on simple port classification removing flows with meaningless values in the 4 counters of the protocol header;
- RTP/RTCP: we rely on the state machine described in [9]. It combines a DPI signature and correlates the value of the fields in consecutive packets (e.g., to check the validity of the counters).
- Emule/BitTorrent: we developed a DPI classifier based on [34, 42] adapting it to the considered scenario.
- Skype: we rely on the Bayesian framework described in [10].

All the aggregated traffic that do not match any of the rules is placed in a sub trace called *Background* since it represents all the *unknown* protocols. Since the oracle itself can be unreliable, accurate manual inspection and pinpointing of suspect cases are detailed in the performance results.

3.3 Results

In this section we report a complete description of the classification performance obtained considering UDP traffic. We start from a simple example, showing the goodness of the SVM decision process with respect to an Euclidean decision process. Then, we present more complex scenarios in which many applications are considered. Finally, we consider the parameter tuning and the resources required by KISS as to operate.

3.3.1 Comparing Euclidean and SVM Decision Process

As described in Sec. 2.4, KISS exploits SVM to model the protocol signatures. However, other solutions are possible, for example using the Euclidean distance, each protocol can be described with an hyper-spheres which identify the area in which the point of an application are expected to fall. The classification process is then straightforward: a point that falls inside a sphere is classified according to the protocol associated to that sphere, while a point that does not fall into any sphere is assumed to be of an *unknown* protocol.

For a given class A, the representative hyper-sphere is fully defined by its center $\bar{X}(A)$ and its radius $\rho(A)$. $\bar{X}(A)$ is simply computed, component by component, as the arithmetic mean of each signature in the training set of class A. The identification of the radius is more complex. Indeed, the hyper-sphere should be big enough to include all the points of the training, but it has to be small enough to avoid to include samples of other classes. Using machine learning terminology, one wants to maximize the *True Positive* ratio while minimizing the *False Positive* ratio.

Formally, the following equation can be used to state the problem:

$$\rho(A) = \arg\max_{\rho} \Big(\%TP(\rho) - \%FP(\rho)\Big) \tag{3.1}$$

Notice that $\%TP(\rho)$ is computed considering samples of class A, while $\%FP(\rho)$ is computed considering samples of all other classes of the training set.

To evaluate the performance of the two decision process, we consider a small subset of the RealTrace-I data set, corresponding to the first 5GBytes of data. Tstat is used to split the trace into 4 sub traces: each sub trace includes only packets classified as belonging to the same protocol, i.e., RTP, Emule, DNS and Background traffic only. Each trace is fed to the KISS classifier, so that signatures are evaluated. Both SVM and Euclidean decision processes are trained using 300 signatures for each class, and the remaining signatures are used to assess the performance of KISS. Recall that a signature is generated every C samples, so that a flow/endpoint can be classified several times (i.e., every C packets).

Table 3.2 summarizes the results. Each row corresponds to a class of traffic according to the oracle. The second column reports the total number of signatures extracted from each sub trace while the remaining columns report the percentages of True Positives and False Positives as defined in Chapter 1.3 for both Euclidean and SVM decision process.

The SVM results are astonishing: the True Positives are always higher than 99% while False Positives are negligible. The performance of the Euclidean classifier are more variable, e.g., it performs very well for RTP but the accuracy decreases when considering Emule and DNS protocols. This is related to the adoption of an hyper-sphere as an approximation of the separation surface between classes. To this extent, Figure 3.1 reports Eq. (3.1) as an example of optimization for RTP, Emule

			KISS-Euclidean		KISS-SVM	
		Tot.	%TP	%FP	%TP	%FP
Tstat	RTP	8386	99.9	-	99.9	-
	Emule	1527	84.3	0.12	99.3	-
	DNS	8245	91.3	-	99.9	0.01
	Backg	2579	99.1	5.31	99.6	0.15

Table 3.2. Comparison of Euclidean and SVM decision process on RealTrace I.

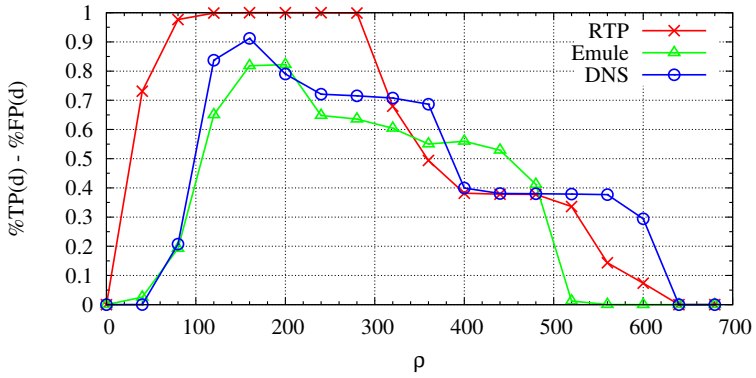


Figure 3.1. Euclidean Decision Process: %TP - %FP for different values of of the hyper-sphere radius ρ.

and DNS. For RTP, any choice of $\rho(RTP) \in [122, 280]$ allows to almost perfectly identify RTP traffic. On the contrary, Emule class is not well represented by the hyper-sphere surface, so that any choice of $\rho(Emule)$ trades between %TP and %FP. Similar reasoning applies for DNS traffic. This shows that a simple decision process based on Euclidean distance is hard to design, while the adoption of SVM allows to avoid this problem. Therefore, in the following we will consider only the SVM classifier and we will investigate how KISS performance are affected by parameters setting and different scenarios.

3.3.2 P2P-TV traffic traces

To prove the KISS flexibility with respect to different applications, we explore its ability to identify traffic generated by P2P-TV applications. The design and engineering of a DPI mechanism for proprietary and closed P2P-TV applications would be daunting and extremely expensive. On the contrary, training KISS is quite straightforward: a packet trace is captured simply running the application, and then it is used to train the SVM. RealTrace-I instead is used as Background class.

	Tot.	Joost	PPLive	SopCast	TVants	Background
Joost	33514	98.1	-	-	-	1.9
PPLive	84452	-	100.0	-	-	-
SopCast	84473	-	-	99.9	-	0.1
TVants	27184	-	-	-	100.0	-
RealTrace I	1.2M	0.3	-	-	-	99.7

Table 3.3. Confusion matrix considering P2P-TV applications.

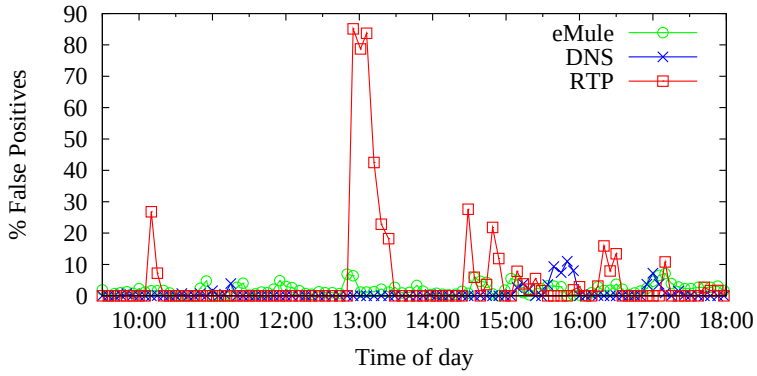


Figure 3.2. False Positive percentage variation versus time. The Background of the classification model do not consider any of the target applications.

In this way, all traces from the NapaTraces data set are used to evaluate %TP while RealTrace-I is instead used to evaluate the %FP, since we assume no P2P-TV traffic could be present during 2006. The total amount of time required to complete this task is less than 6 hours.

Results are summarized in Table 3.3, which reports percentages computed over more that 1.2 millions of tests. Labels on the lines represents the ground truth. Also in this case, results are amazing: KISS is able to correctly classify more than 98.1% of samples as True Positives in the worst case, and only 0.3% of False Positives are present.

3.3.3 Signature Robustness

We are first interested in quantifying KISS robustness with respect to a training set independent from the test set. We thus perform an experiment in which the SVM is trained using samples extracted from the initial part of the RealTrace-I. A 9 hour long subset of RealTrace-I is considered, but the training set includes samples extracted from the first 30 minutes only. As in the experiment described in Sec. 3.3.1,

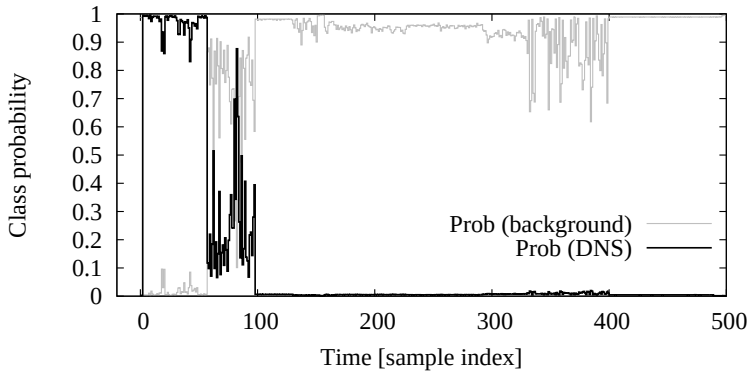


Figure 3.3. Example of an endpoint that causes False Positives. Different classification windows over time.

only RTP, Emule, DNS are considered as known classes while the remaining traffic is included in the Background class. Results are reported in Figure 3.2 showing only Background False Positive percentages since the %TP is always higher than 99%. The plot confirms the intuition that the characterization of the Background traffic may be a problem, since there are peaks that clearly show that the SVM is fooled by the sudden appearance of unknown protocols that were not included in the training set.

Investigating further, we notice that the high percentage of Background traffic classified as RTP traffic is due a single endpoint which is receiving traffic with the same "format" of RTP protocol. However, the DPI based oracle did not classify this endpoint as RTP, since a mismatch in the RTP field is present: it takes a values of 1 instead of 2. Apart from this difference, all other fields are in perfect agreement with the RTP standard. Moreover, all packets received by this endpoint have 172B of UDP payload, which is typical of VoIP streams using the ITU-T G.711 encoder [9] used in the FastWeb network. We then claim that this is an *actual* RTP flow, but the DPI oracle was fooled by the wrong version value. On the contrary, KISS correctly classifies this flow as a RTP flow.

Similarly, investigating the samples that are misclassified as DNS (e.g., from 15:30 to 16:00) we notice that a single endpoint (listening to port number 9940) is the only responsible for this behavior. We manually inspected this traffic, and verified that it cannot be a DNS endpoint, so that the oracle is reliable. Interestingly, no sample of this endpoint is included in the training set of Background traffic. Since the SVM is always forced to classify the sample as one of the possible classes, it resolves to classify it as DNS rather than Background. Considering this endpoint only, Figure 3.3 shows the probability that the SVM evaluates it as a Background or DNS sample versus time. It can be seen that some uncertainty is present. Repeating

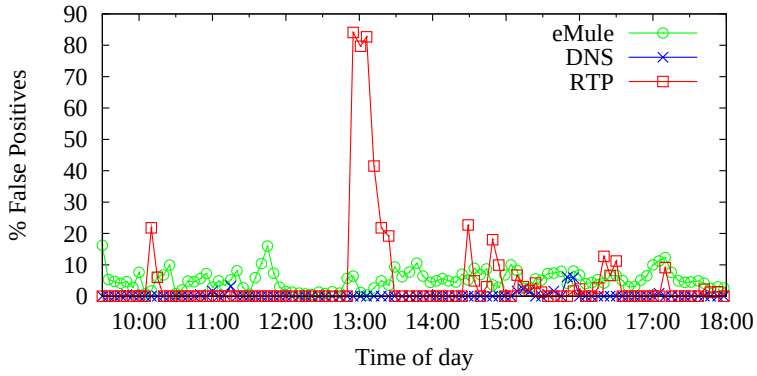


Figure 3.4. False Positive percentage variation versus time. The Background of the classification model contains also traffic of the target classes.

the experiment by including some of these endpoint signatures in the Background training set, KISS correctly classifies it. This is an example of "under-training" of SVM.

Similar conclusions can be drawn investigating the Emule False Positives. They all correspond to endpoints listening to port number 3074, possibly related to the Xbox-Live protocol, which is sometimes confused by the SVM as Emule traffic, since the SVM is "under-trained". Also in this case, by adding some samples of these endpoints to the training set, no more FP is detected.

We can conclude that KISS shows excellent performance, since in all cases the True Positive percentages are higher than 99%. The training of the SVM is robust considering the signature of known protocols, but it can suffer when the Background training set is small or does not include all protocols that may be present in the considered network scenario. This leaves room for improving the performance of KISS by carefully selecting the training set samples. Notice that the accuracy of any supervised machine learning decision process is strongly affected by the coverage and accuracy of the training set. Intuitively, a limited or outdated training set performs worse than an updated one.

3.3.4 Training with the Aggregate

A possible weakness of KISS is that the SVM must be trained with the Background traffic, i.e., with actual traffic extracted from the network the classifier is used representing the *unknown* protocols. While the adoption of actual traffic does not pose particular issues, the extraction of "pure" Background is very questionable. A possible solution to this issue is to use, during the SVM learning phase, the whole aggregate of traffic as unknown traffic. This poses some problems, since samples of

	Tot.	BitT.	Emule	RTCP	RTP	DNS	Skype	SopCast	TVants	PPLive	Backg
BitT.	1268	100	-	-	-	-	-	-	-	-	-
Emule	57255	0.02	99.15	-	-	0.03	-	-	-	-	0.80
RTCP	2407	-	-	99.96	-	-	-	-	-	-	0.04
RTP	585647	-	-	-	99.79	-	-	-	-	-	0.21
DNS	2707	0.46	-	-	-	99.54	-	-	-	-	-
Skype	46600	-	-	-	-	-	99.61	-	-	-	0.39
SopCast	83460	-	-	-	-	-	-	99.95	-	-	0.05
TVants	25748	-	-	-	-	-	-	-	99.69	-	0.73
PPLive	27278	-	-	-	-	-	-	-	-	99.24	0.76
Backg	84273	0.27	7.59	-	2.67	0.22	-	-	-	-	89.25

Table 3.4. Confusion matrix considering P2P-TV, Real-Traces and Skype traffic.

a given class may be part of the aggregate traffic as well.

Figure 3.4 shows results obtained by running KISS in the scenario previously described, but using the aggregate trace to train the SVM for the unknown traffic. Also in this case the True Positive percentage remains higher than 99% (results are not plotted for the sake of brevity). Considering FP, apart from the RTP endpoint that the oracle misclassifies, we observe an increased percentage of samples being classified as Emule (with an average %FP=4.5%). Nonetheless, results are very good.

3.3.5 Training with many classes

All the results reported so far consider only 3 or 4 protocols. It is interesting to analyze the performance of the classifier with a larger number of target protocols. Using RealTrace-II, P2P-TV testbed and the Skype data sets we create a KISS model including nine different classes, plus one for the Background traffic. Each class has been characterized with 300 signatures randomly chosen from the initial portion of each trace. Table 3.4 reports the confusion matrix of the classification result. As before, labels on the lines represents the ground truth. The first column reports the total number of signature, while the other columns show the agreement between the ground truth and KISS. Again, results are impressive: KISS always achieves more than 99% of True Positives, with less than 10% of False Positives from the background class. Further analysis revealed that 7.59% of the false Emule samples are related to a single endpoint, which generates lots of short flows directed to an high number of different destinations. Unfortunately, we were not able to identify which actual protocol was used. After the adding of some samples of this endpoint in the background training set, all Emule False Positives disappeared. For what concern the 2.67% of samples identified as RTP, more than the 90% of them is generated by only two endpoints that use a RTP protocol with wrong version

number as previously discussed in Sec. 3.3.3.

3.4 Parameters selection and Tuning

The signature creation approach previously presented is based on a number of parameters whose setting may be critical. These are the criteria we used to set them:

Bits per group, $b = 4$. The choice of b should trade-off two opposite needs. From the one hand, we would like b to be the closest as possible to typical length of protocol fields; since protocols dialogs are usually based on words whose length is multiple of the byte or, sometimes, is half of a byte, b should be 4 or 8 or a multiple of 8. From the other hand, b should be small enough to allow that the packet window C over which the Chi-Square test is statistically significant is not too large, so that streams can be classified even if they are not too long, they are classified in short time and live classification is possible. Thus, we chose $b = 4$.

Number of bytes per packet, $N = 12$. In general, classification accuracy increases with the number of considered bytes per packet. However, complexity of the classification tool increases also with the N, in terms of both memory and computational complexity. As a convenient trade-off we choose $N = 12$. Given $b = 4$ this values corresponds to $G = 24$ groups. Another reason to choose $N = 12$ bytes is that, this way, we collect 20 bytes of the IP packet payload (12 bytes + 8 bytes of the UDP header) that is the minimum size of the TCP header and the typical value used by measurement tools. Notice that the optimal value of N depends on the targeted applications. For example, DNS and Emule can be clearly identified by only considering (X_2,X_3) (see Figure 2.3). However, when considering different protocols, possibly more and different groups must be considered.

Packet window, $C = 80$. While we would like to keep the packet window as small as possible, the estimation of the observed distribution is considered to be statistically significant if the number of samples for each value is at least 5. Having chosen $b = 4$, in order to have $E_i = C/2^b$ equal to 5, we need C to be equal to about 80.

However, since in KISS we are not performing a real Chi-square test, we are interested in the impact of smaller values of C, which would allow an earlier classification. Figure 3.5 reports the True Positive percentages of well-known protocols, and the False Positive percentages, without distinguishing among protocols. Confidence intervals with a confidence level of 5% are evaluated over 250 different sub-traces from RealTrace I, each comprising more than 100 samples. The Figure clearly shows that

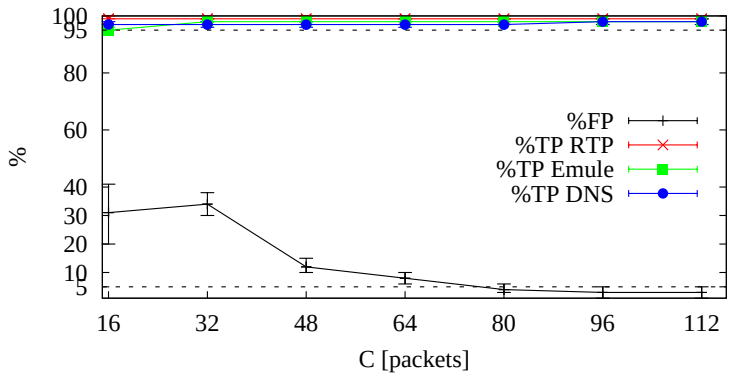


Figure 3.5. Classification accuracy versus signature packets window C.

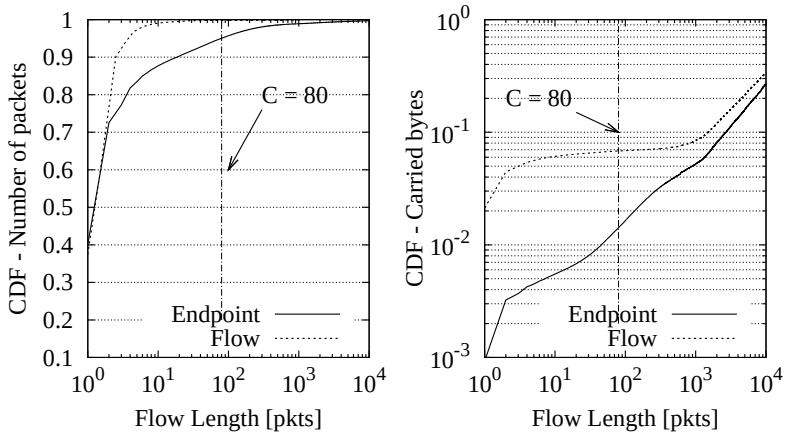


Figure 3.6. CDF of the flow/endpoints length in packets (on the left), and bytes (on the right). The vertical line is in correspondence of 80 packets.

the %TP is almost not affected by the number of samples that are considered to evaluate the observed frequencies in Eq. (2.1). Indeed, the format of the considered protocols is very different and the SVM has little problems in distinguishing them even if C is small. However, the %FP is much more sensitive to C, and only for $C > 80$ it goes below 5%.

3.4.1 Coverage

The packet window size C plays an important role in the KISS design, and it may affects the applicability of KISS. Indeed, given the connectionless characteristic of UDP, one expects that UDP flows and endpoints last for few packets. Figure 3.6

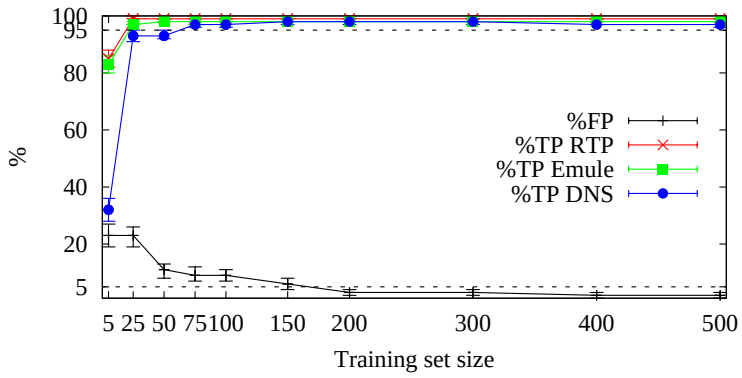


Figure 3.7. Classification accuracy versus training set size.

confirms this intuition reporting the Cumulative Distribution Function (CDF) of flow length for both flow/endpoint packets and bytes. All incoming UDP traffic in RealTrace-I is considered to derive the CDF. The left plot clearly shows that 40% of flows and endpoints has only 1 packet, while only 0.2% of flows and 5% of endpoints have at least 80 packets. However, these flows/endpoints respectively account for more than 93.8% and 98.6% of the *bytes* carried by UDP, as reported in the right plot. This clearly shows that, while KISS is not suitable for the classification of short lived UDP flows/endpoints, it can however successfully target the small fraction of them that generate the majority of the traffic, i.e., long-lived flows.

3.4.2 Trainset set size

It is interesting to observe how performance changes with training sets of different size. Results are plotted in Figure 3.7, which reports the %TP and %FP for increasing training set size with confidence intervals evaluated over 10 tests with a confidence level of 5%. The plot shows that KISS classifies RTP, DNS and Emule correctly even with only 25 samples (worst case is %TP>91.73% for DNS) but at least 75 samples are needed to have excellent results. Also in this case, more problematic is the correct classification of the Background traffic, since the False Positive percentage goes below 5% only when the training set comprises at least 200 samples. The intuition behind this is that the Background traffic is far more heterogeneous with respect to traffic of a given protocol, and a larger number of samples are required to accurately describe it.

Packet processing	Feature Extraction

EP_state = hash(IP_d, $port_d$)
for g = 1 to G **do**
  P_g = payload[g]
  EP_state.O[g][P_g]++
end for
	E = C/2^b *(precomputed)*
	for g = 1 to G **do**
	  Chi[g] = 0
	  for i = 0 to 2^b **do**
	  Chi[g] += (EP_state.O[g][i]-E)^2
	  end for
	  Chi[g] /= E
	end for

Table 3.5. Pseudo algorithm for packet processing (left) and feature extraction (right).

Cost	Analytical	Example ($G = 24$,$b = 4$)
Memory allocation	$G \cdot 2^b$ counters	384 bytes
Packet processing	$(2G + 1)lup + Gsim$	$49\,lup + 24\,sim$
Feature selection	$2^{b+1}\mathrm{G}\,lup + \mathrm{G}\,com + (3{\cdot}2^b{+}1)\mathrm{G}\,sim$	$768\,lup + 24\,com + 1176\,sim$

Table 3.6. Numerical case study of the resource requirements of the classifiers.

3.5 Complexity

Finally, in this section we investigate on the resource requirements of KISS. We start presenting a functional analysis, investigating on costs involved in the packet processing and protocol signatures creation. Then we present some results on the classification responsiveness and memory consumption obtained analysing packet level traces.

3.5.1 Functional analysis

To provide an analysis of the resource requirement of KISS, we follow a theoretical approach calculating the memory occupation and the computational processing from the formal algorithm specification. In this way, our evaluation is independent from specific hardware platforms or code optimizations. Table 3.5 reports the pseudo algorithms for packet processing (left) and feature extraction (right). Instead, Table 3.6 summarizes both the theoretical the numerical quantification of the costs considering the default configuration of the classifier.

KISS framework extracts application protocol fingerprints applying statistical

analysis on the packet payload (see Chapter 2.3). For each new packet, the classifier retrieve the data structure related to that endpoint (flow) performing an hash operation using IPs and ports. Then, the first bytes of the payload are divided in g groups of bits and the associated values are stored in the endpoint (flow) data structure. When enough statistics have been collected a new signature is computed accessing to the whole endpoint (flow) data structure and applying Eq. (2.2) also reported in the following:

$$X_g = \sum_{i=0}^{2^b-1} \frac{\left(O_i^{(\mathrm{g})} - E_i^{(g)}\right)^2}{E_i^{(g)}} \tag{3.2}$$

where $E_i^{(g)} = C/2^b$.

Memory footprint is mainly related to the data structures used to compute the statistics. KISS requires a table of $G \cdot 2^b$ counters for each endpoint to collect the observed frequencies employed in the chi-square computation. Considering the default parameters (i.e., $G = 24$ chunks of $b = 4$ bits) each endpoint requires 384 counters. A new signature is issued after a window of 80 packets (see Chapter 3.4 and Sec. 4.3) so each counter can be stored in a single byte and each endpoint (flow) needs 384 bytes to store the statistics.

Instead, the computational cost are computed considering three classes of operations: *lup* for memory lookup operations, *com* for complex operations (i.e., floating point computations), *sim* for simple operations (i.e., integer computations). Considering packet processing, memory lookup operations represent a critical cost for KISS since we need to access to the packet payload without affecting the network card speed while operating the classification in real time. In particular, KISS performs $2G + 1 = 49$ lookup operations for each packet, half of which are accesses to packet payload. Then G integer increments are performed. Instead, the signature extraction process requires to access to the whole data structure of an endpoint (flow) for a total of 768 lookup operations and to perform 24 floating point and 1176 integer operations.

3.5.2 Example of resource consumption

As to better express the costs of the classification technique proposed, we performed a simple example. We consider RealTrace-I, RealTrace-II and the NapaTraces data set reported in Table 3.1. We create a classification model as described in the previously described and we measure the actual resource consumption classifying the packet traces on a common PC (Intel(R) Core(TM)2 Duo CPU T8300 @ 2.40GHz with 4GB of Ram).

Overall, the traces are processed in less than 2 minutes with an average memory consumption of 3MB. In Figure 3.8 we show the CDF of the amount of time needed

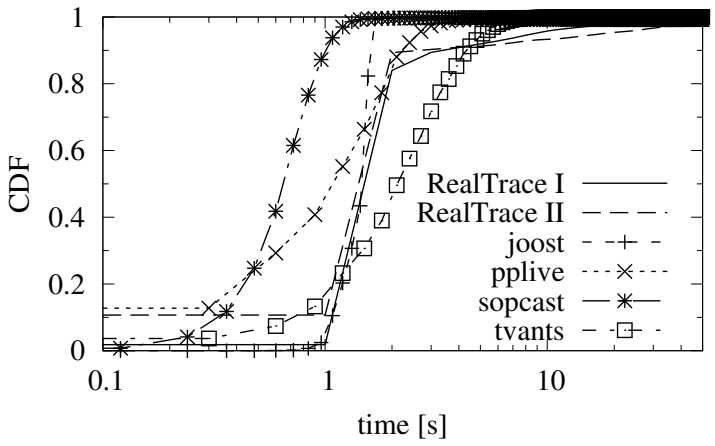


Figure 3.8. Cumulative distribution function of time required by KISS to generate a new signature.

to collect 80 packets for an endpoint. It can be observed that for the P2P-TV applications a new signature is computed every 2 seconds while for the aggregated traffic in RealTrace-I and RealTrace-II 90% of the endpoints need to wait less than 10 seconds.

The results obtained testify the goodness of the proposed technique and leave space for further improvements through optimizations. For example, garbage collections can be introduced as to better control the memory allocation while to I/O primitives can be reduced as to speedup the classification process.

Chapter 4

Evaluating KISS on TCP traffic

In this chapter we report the evaluation of KISS on TCP traffic. Recalling that the classification overflow is the same for both TCP and UDP, (see Chapter 2.2), the main difference in adopting TCP instead of UDP is on the process used to extract the protocol signatures: only endpoints are considered, aggregating the first P packets of F different flows.

In the remaining of this chapter, we start introducing the data set used. Then, we present the classification results obtained and we conclude with a description of the parameter tuning.

4.1 Data set

To evaluate KISS on TCP traffic we consider a data set collected during May 2008 at the egress router of Politecnico di Torino network. The traces correspond to a one week long dataset, in which about 7000 internal hosts exchange data with more than 3 million different hosts in the Internet. Details concerning the traffic volume, in terms of number bytes, packets, flows, endpoints and signatures, are given in Table 4.1. The table reports the total traffic volume, and the breakdown across the most common application protocols considered, namely HTTP, FTP, IMAP, POP, Skype, SMTP, SSH while the remaining are aggregated in the class "Other".

We focus only on *internal endpoints*, i.e., servers whose IP address is internal to the campus LAN (see Chapter 2.3.2). Moreover, we need to observe several flows involving a single endpoint to gather a single signature, and thus take a classification decision. In case of external endpoints, this means that several of our internal hosts have to contact the same endpoint to collect enough packets to compute the signature. While this is not an issue for popular external server and protocols (e.g., popular Web sites), however it limits the number of protocols we could use considering the dataset we use in this paper.

Protocol	Bytes (GB)	Packets (M)	Flows (k)	Endpoints	Signatures
HTTP	343.67	507.08	6531.19	177	114222
FTP	0.04	0.65	19.39	21	229
IMAP	0.73	1.34	2.49	10	66
POP	3.40	7.74	156.39	25	3551
Skype	1.95	20.38	145.22	322	2752
SMTP	61.00	126.61	4917.20	56	83677
SSH	8.84	19.47	31.64	141	304
Other	453.83	744.53	13400.98	1512	46773
Total	873.46	1427.80	25204.5	2246	251574

Table 4.1. TCP data set collected at Polito campus network.

Our dataset includes more than 250000 signatures, that refer to about 2250 endpoints. As expected, Web service constitutes the bulk of traffic, while a fairly large amount of incoming SMTP traffic is present. The protocols we consider account for more than 95% of the traffic (in terms of bytes, packets and flows), yielding to a large fraction of the traffic to be labeled as "others", which therefore includes other unknown protocols.

Concerning the number of available signatures, notice that each internal endpoint has to be contacted by F different hosts of at least P packets to compute the signature. The number of signatures per protocol depends on the arrival pattern as well as on the flow length as well.

4.1.1 The oracle definition

As already pointed out in [53], the definition of a reliable DPI oracle is a daunting task, that we have to carry on due to the lack of a labeled dataset. Except for the Skype protocol, for which we resort to [10], we devise a two-stages DPI oracle, defined as follows.

- **Port filter**: we start considering TCP port number, focusing only on those flows whose TCP destination port corresponds to well-known services, i.e., 80 for HTTP, 22 for SSH, and so on. By doing so, we forcibly miss some endpoint. For example, HTTP servers running on port 8080 or on other non-standard ports end-up in the "other" protocol sub-trace. However, this choice yields to a conservative evaluation of the classification performance results;

- **Protocol syntax check**: the second phase involves application protocol check, that is done using the Wireshark tool. Wireshark is a well-know sniffer

Dst	HTTP	FTP	SMTP	IMAP	Skype	SSH	POP	Other
HTTP	94.94	0	0.06	2.58	0	0	0	2.39
FTP	0	98.59	0	0	0	0	0.03	0
SMTP	0	0	99.86	0	0	0	0	0
IMAP	0.02	0	0	90.97	0	0	0	0
Skype	0.01	0	0	0	100	0	0	0.05
SSH	0.05	0	0	0	0	100	0	0.03
POP	0.01	1.31	0.02	2.9	0	0	99.94	0
Other	4.97	0.1	0.06	3.55	0	0	0.03	97.53

Src	HTTP	FTP	SMTP	IMAP	Skype	SSH	POP	Other
HTTP	91.63	0	0.07	1.54	0	0	0	13.99
FTP	0.35	98.98	0.02	0	0	0	0	1.05
SMTP	0	0.03	99.45	0	0	0	0.03	0.03
IMAP	0	0	0	58.08	0	0	0	0
Skype	0.01	0	0	0	100	0	0	0.03
SSH	0.15	0	0	0	0	100	0	0.05
POP	0	0	0	0	0	0	99.59	0
Other	7.86	0.99	0.46	40.38	0	0	0.38	84.85

Table 4.2. Classification performance for traffic directed to (top) or originated from (bottom) the server-side endpoint.

which is able to *parse* the headers of known protocols. In case during the parsing Wireshark fails to identify the protocol, the flow is moved to a sub-trace containing all the other protocols.

4.2 Performance evaluation

Evaluation of classification performance is conducted over the entire dataset, by comparing the SVM labels to the DPI oracle labels for each signature. Results reported in this section refer to a test in which the training set containing 5000 signatures, proportionally balanced across protocols. Instead, 1800 training signature are used to describe the "other" protocol set, since this set comprises possibly several protocols and its proper description requires that such protocols are well represented in the training set. A sensitivity analysis to the training set size is not reported due to lack of space. Readers are referred to [44], which shows that, even considering only 35 signatures per each of the known protocols the classification results are minimally compromised. This is a consequence of the discriminative power

of SVM, whose performance are known to be highly robust even in presence of few learning samples.

Each test is repeated 10 times, by randomizing the training set at each execution, and validating the model on the remaining signatures. Average results over all 10 iterations are reported in Table 4.2. Each column corresponds to a sub-trace filtered by the Oracle, which is fed to a trained SVM, whose output labels are reported on each row. Thus, diagonal elements of the confusion matrix account for True Positive classification (i.e., a protocol labeled as X by DPI is also labeled as X by SVM). Conversely, cells outside the diagonal refer to misclassified signatures: a protocol labeled as X by DPI is labeled as Y by SVM; this decision accounts for False Positive classification of Y and False Negative classification of X.

Results considering the two different traffic directions are reported. Top (bottom) portion of the table reports the case where traffic is destined to (originated from) the internal server endpoints. Notice that, although classification results are very good in both cases, best results are obtained when traffic is destined to the server endpoints. This is visible for HTTP, IMAP and Other protocols. The intuition behind this is that the client protocol requests are easier to characterize than the server replies, which can be more variable. For example, HTTP requests use limited set of protocol keyword as discussed in Sec. 2.3.2, while server answers can be much more different.

Focusing on traffic destined to the server, we gather that true positive rate classification always exceeds 90.97%, with an average of about 97.62%. Compared to the UDP classification results presented in the previous chapter which yielded a 98% true positive rate in the worst case, the classification performance of TCP traffic decreases. This is somehow expected: in the UDP case, application protocol headers are present in each segment, yielding to very reliable KISS signatures; in the TCP case, the TCP connection oriented service and segmentation algorithms affect the KISS signatures, that are possibly computed over both application protocol headers and actual data carried in the first 5 TCP flow segments.

4.3 Parameter Selection and Tuning

As for UDP Traffic, TCP signatures depend on a number of parameters, some of which (b,C and N) pertain to the X_g metric while the remaining (P and F) are related to the description of the TCP endpoints. For the first group of parameters are still valid the considerations reported while considering UDP traffic, i.e., $b = 4$ (chunks of 4 bits), $N = 12$ (consider the first 12 bytes of the payload) and $C = 80$ (80 packets to extract the χ^2 from each chunk).

For the remaining parameters are valid the following observations:

Number of packets per flow ($P = 5$). The segmentation imposed by TCP yields

an upper bound on P, the maximum number of packets at the beginning of a flow carrying application header at the beginning of the payload. As far as the number of packets per flow is concerned, we employ $P = 5$ which was observed to be a good value in [6, 16]. Sensitivity analysis to P is provided in [44].

Number of flows per endpoint (F=C/P=16). Constraints on C and P yield a lower bound on $F = 16$, the minimum number of flows to observe before an endpoint classification decision can be taken. This translates into a constraint on the classification timeliness, i.e., how fast and frequently the classification can be taken, since the start of F different flows have to be observed prior that a classification decision can be taken. Notice however that, the more active the endpoint, the quicker the identification (which is beneficial since operators are interested in classifying volumes of traffic, and should pose no problem in discriminating between active endpoints such as server vs P2P).

Chapter 5

Comparing P2P-TV Traffic Classifiers

Internet Television is perhaps the most important emerging application in these days. While services based on traditional client-server technologies such as Hulu, Miro or (in a broader sense) YouTube are currently widely used, new applications are emerging that exploit the peer-to-peer (P2P) paradigm to broadcast television over the Internet. However, network operators fear the potential impact that these applications may have on the network, since they can offer a significant load on the system, which can cause network congestion and possibly collapse, with the failure of the P2P-TV service itself [46]. Therefore, there is lot of interest in new techniques capable to monitor the complex traffic patterns generated by these systems, which unfortunately use proprietary and undocumented protocols, and are therefore harder to identify than their open standard counterparts.

In Chapter 3 we already proven the accuracy of KISS with respect to P2P-TV traffic. In this chapter we extend this analysis comparing KISS with other two traffic classifiers focusing on on UDP traffic, which is currently the most part of the traffic generated by P2P-TV applications. pDPI [63], the first classifier we consider, is based on a traditional per-packet Deep Packet Inspection (DPI) which identifies the traffic according to a set of application-layer regular expressions. IPSVM [23] instead is a statistical classifier using feature extracted from the transport-layer headers.

The three classifiers are tested considering a common data set of traces that include ground truth information, captured using either the GT suite [29] or setting up experiments in controlled environments. This allows us to test the classification accuracy of each technique, and its robustness with respect to different protocols, trace capture time and location. Overall, the three approaches show strengths and weaknesses in different areas.

The rest of this chapter is structured as follows. We start giving a brief description of pDPI and IPSVM. Then, we present the methodology used to collect the

	pDPI	IPSVM	KISS
Technique	Deep Packet Inspection	Transport layer statistics	Stochastic Packet Inspection
Feature	Regular expressions on L7 payload	Length of packets	Stochastic description of L7 payload
Aggregation	Flow	Flow	Endpoint
Decision	Pattern matching	SVM	SVM
Training	Manual, difficult	Automatic	Automatic
Support Encryption	No	Yes	No
Pros	Well accepted Higher completeness	Easy training	Easy training; High true positives
Cons	Require application protocol reverse engineering	Higher false negatives	Require training for the "unknown" class

Table 5.1. Main properties of the classifiers.

traffic describing the data collected. Finally, we conclude comparing the results and detailing the strength and weakness of each technique.

5.1 Traffic Classification Techniques

Since the three classifiers we are considering exploit different approaches, in this section we summarize the key elements required to understand their operations. For a complete description of pDPI and IPSVM we refer the reader to [63, 23].

Table 5.1 reports an overview of the main properties of the three classifiers. We start observing that they operate on different classification objects (see Chapter 1.1.1): pDPI and IPSVM operate on a per-flow basis, while for this work KISS operates on endpoints. As introduced in Chapter 2, KISS can classify both flows and endpoints. However, in this work we show the benefits of using the endpoints as classification object as to achieve a better coverage of the traffic classified. Given the different type of classification object, KISS applies a simple strategy to propagate the endpoint labels at flow level: given a flow, when the same label is associated to both the source and destination endpoints, the flow is tagged with such label. In case a different label is associated to the two endpoints, the flow is marked as unknown (implementing then a conservative algorithm).

5.1.1 pDPI

A DPI classifier relies on the observation that each application uses specific protocol headers to initiate and/or control the data exchange, which can be captured by a signature usually corresponding to a regular expression.

Among the several flavors of DPI classifiers, the most common implementations operate either on packets or on application-layer messages. While the message-based variant appears to be more precise (and in fact it is extensively used in network security applications), it is more complex because it may require an additional phase of segments reassembly and IP de-fragmentation to rebuild the original application-layer message. Furthermore, some studies [53, 63] suggest that its results are roughly equivalent to the ones obtained by a simpler packet-based approach in case of regular traffic. For this reason this paper considers a traditional packet-based DPI classifier (in short *pDPI*).

5.1.2 IPSVM

IPSVM [23] is based on a single-class SVM as proposed in [68]. This classifier requires a training phase to automatically derive statistical models, one for each application, which are then used during the classification phase.

The general workflow of this classifier is similar to the one used by KISS but there are two important differences. First of all, the feature extraction process exploits the packet dimension instead of a Stochastic Packet Inspection. Given a flow, the classifier extracts a vector $X = (x_1,x_2,...,x_d)$ of d values corresponding to the length of the UDP payload of the first d packets. Positives and negatives values are used to represent the direction of communication. During the training phase, for each target application, a given number v of vectors X is fed to a SVM machine, obtaining a set of binary models. The optimal values of d and v are automatically calculated in the training phase. For the work presented in this chapter, they were set to $d = 2$ and $v = 1000$.

Instead, during the classification phase, a vector X is fed to the generated models and a rejection threshold (specific for each model and derived during the training phase) is used to access the optimal classification. If the similarity index of the classification falls below all the thresholds, then the sample is assigned to the *unknown* class. Differently from KISS, IPSVM do not need "Background" traffic during the training phase.

5.2 Data sets

In this work we evaluate three of the most widespread P2P-TV applications as benchmarking data set: PPLive, SopCast and Tvants. We focus only on UDP traffic since most of P2P-TV traffic relies on UDP, e.g., in our traces UDP accounts for 88%, 100% and 30% of bytes for PPLive, SopCast and Tvants respectively.

We compared the three classifiers using four data sets of traffic traces, whose main characteristics are summarized in Tab. 5.2.

Name	Duration	Flows	Bytes	Endps
VMware (VM)	75 h	6.1 M	65.5 G	606 k
napa-IT (N1)	3 h	68 k	1 G	19 k
napa-PL (N2)	3 h	61 k	1 G	17 k
operator (OP)	10 m	101 k	478 M	17 k

Table 5.2. Data sets used in the experiments (UDP traffic only).

Software	Napa-Wine	VMware
PPlive	1.9.21	2.16
Tvants	1.0.0.59	1.1
SopCast	3.0.3	3.0.3

Table 5.3. P2P-TV data sets software versions.

VMware (VM): this data sets was collected between Dec. 2008 and Jan. 2009 using a set of VMware virtual machines running Windows XP on which the three target applications were scheduled synchronously: every hour all the machines run the same application tuned on the same TV channel for 45 minutes followed by 15 minutes of silence to purge all opened connections. Using this approach, we obtained 8 captures per application every day. In addition, all the virtual machines had the Skype client turned on, while Emule and BitTorrent clients (with protocol obfuscation enabled) were active only on one machine. The virtual machines were installed on a single server inside the Politecnico di Torino network while the aggregated traces were collected at the border gateway that connects the campus LAN to the Internet.

napa-IT (N1) and napa-PL (N2): these traces have been collected on the 4th of April, 2008 at the Politecnico di Torino and Warsaw University of Technology respectively, in the context of the Napa-Wine project [46]. Napa-Wine is an European project in which large scale experiments are periodically organized to study the behavior of the P2P-TV applications. Similar to the previous case, several (real) Windows XP hosts were used to run each P2P-TV applications in isolation, and to collect traffic.

operator: this trace contains real traffic collected in May 2006 at a large ISP PoP located in Turin, Italy[1]. The PoP aggregates more than 2000 users using different technologies, e.g., VoIP phones, set-top-boxes, PCs, etc., and running any applications, e.g., file sharing applications, web browsing, gaming, viruses,

[1]The name of the ISP is not reported for privacy reasons.

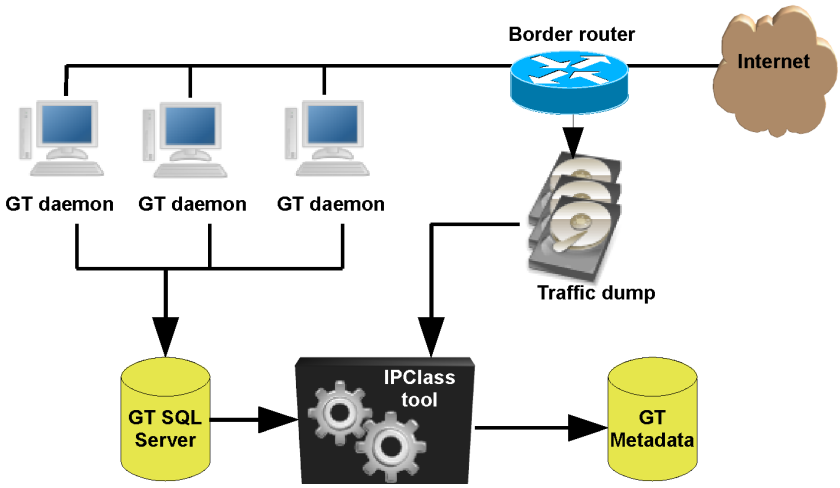


Figure 5.1. The GT architecture.

etc. Since at that time P2P-TV applications were not popular at all in Italy, these traces do not contain P2P-TV traffic and hence they can be used to analyze the capabilities of the classifiers in terms of false positives. We verified this assumption by a manual inspection of the traces.

Since the VMware and Napa-Wine experiments are spaced by eight months and both PPLive and Tvants automatically update theirs clients to the last released version, different versions of the software (and possibly protocols) are present in the data sets. This characteristic was used to determine the robustness of the various techniques with respect to an application update. Table 5.3 reports the precise application version number used. As we can see, in the VMware data set PPLive and Tvants has been updated with respect to Napa-Wine data sets.

5.2.1 The oracle definition

The availability of the ground truth for each data set is perhaps the most critical problem when analyzing traffic classifiers [53]. The ground truth for the *VM* traces has been derived using the GT framework [29, 30], which is a software that associates accurate ground truth information to traffic traces. Figure 5.1 depicts the configuration schema used to extract the ground truth. It requires three components: a *GT daemon* running on each monitored host a *SQL database* and the splitting tool `ipclass` [30]. The daemon running on the host is able to identify the creation/destruction of a socket and associates it to the application that has generated it. The identification process is not synchronous and each 100 ms the demon polls

	VM		napa-IT (N1)		napa-PL (N2)	
	flows	bytes	flows	bytes	flows	bytes
PPlive	1.0 M	24.5 G	27 k	168 M	30 k	592 M
Tvants	3.5 k	1.1 G	8 k	249 M	8 k	189 M
SopCast	201 k	30.4 G	34 k	626 M	24 k	297 M
Emule	183 k	55 M	-	-	-	-
Skype	4.1 M	2.4 G	-	-	-	-
BitTorrent	33 k	14 M	-	-	-	-
nolabel	422 k	8.1 G	-	-	-	-
summary	6.1 M	65.5 G	69 k	1.0 G	62 k	1.1 G

Table 5.4. Data sets ground truth.

the system looking from modifications in the socket table[2]. The collected data (IP addresses, UDP ports, application name, and some timestamps) are then stored into a *SQL database* while a packet capture engine co-located with the network's border gateway creates dump traces at the same time. When the capture is completed, using the information stored in the SQL database the `ipclass` [30] tool splits the aggregated traffic in sub-traces with respect to the applications.

Since the `GT` framework was not yet available when the Napa-Wine traces were captured, the ground truth for these traces has been obtained by removing all the standard protocols (DNS, NTP, ...) from the traces. Since each machine had no daemons (except for unavoidable systems services) or other applications running in background and a single P2P-TV application was executed at each time, all remaining traffic has been assumed as generated by the target applications. We verified this speculation by manually inspecting a set of randomly selected flows and by further analyzing this trace with a pDPI classifier.

Table 5.4 details the result of the splitting of the traces containing P2P-TV traffic. The *operator* traces instead, since they do not have any P2P-TV traffic, are used as *unknown* to check the accuracy of the classifiers with respect to other type of traffic.

5.3 Classifiers comparison

We compare the traffic classifiers considering two different aspects: *coverage* and *accuracy*. In the first case we analyze how much traffic the classifier is able to elaborate, while in the second case we assess the accuracy of the classification. Both these

[2] Due to the polling mechanism used by `GT`, a limited amount of traffic (7% of flows and the 12% of bytes) is not labelled and is therefore excluded from our analysis.

		IPSVM		KISS	
		%Flows	%Bytes	%Flows	%Bytes
VMware	PPlive	47.6	0.3	0.2	-
	Tvants	1.3	-	-	-
	SopCast	3.0	-	0.1	-
napa-IT	PPlive	76.7	2.4	0.2	-
	Tvants	1.2	-	1.3	-
	SopCast	3.6	-	1.2	-
napa-PL	PPlive	55.5	0.7	-	-
	Tvants	0.2	-	2.9	0.1
	SopCast	1.7	-	-	-
operator	other	33.6	0.8	9.4	0.7
	TOTAL	**7.1**	**0.1**	**0.4**	**-**

Table 5.5. Traffic that IPSVM and KISS cannot classify.

analysis are obtained using `diffinder` [14], a tool that we created which is capable of generate either aggregated statistics (e.g. number of concordance/discordance between two classifiers) or produce the complete list of flows classified differently for further analysis. Since the classifiers may have different granularity (e.g., a DPI can differentiate between HTTP 1.0 and 1.1) and in general the name of the application-layer protocol may be different in the various classifiers, the tool uses a mapping file to compare the results.

Since all the classifiers require some tuning to properly address the target application, we used part of the collected data sets for this purpose. In particular, the P2P-TV signatures used by pDPI have been derived by reverse engineering the Napa-Wine traces (the oldest with respect to applications' versions). Vice versa, the statistical model for IPSVM and KISS has been generated using the VMware data sets.

5.3.1 Coverage

While the pDPI classifier is potentially able to classify a flow by inspecting the first packet, IPSVM and KISS may require a larger number of packets. In fact, for each flow, IPSVM uses the features extracted from the first d packets of the flow itself, with $d = 2$ in our case. Consequently, all flows lasting only one packet cannot be classified. Instead, KISS needs 80 packets per each endpoint to create its signature. If on the one hand endpoints generating less then 80 packets cannot be classified, on the other hand, once properly trained, KISS can classify incoming

flows as soon as it analyzes the first packet, provided that the associated endpoint is already known. Moreover, KISS can classify established sessions, while both pDPI and IPSVM cannot because they analyze the first packets of the session.

Table 5.5 reports the amount of traffic (in terms of flows and bytes) that remains unclassified in IPSVM and KISS because of these limitations. While the percentage of unclassifiable traffic is definitely limited with both techniques, it is interesting to note that PPLive has a large amount of short sessions that last only one packet, which generate a fair high number of unclassifiable flows with IPSVM but whose impact in terms of bytes is negligible. This does not affect KISS thanks to its peculiar endpoint aggregation, although the constraint of having 80 packets seems to be more sensible in case of the *operator* trace, which contains many endpoints generating less than 80 packets. Also in this case, however, the impact in terms of bytes is definitely limited.

5.3.2 Accuracy

Table 5.6, Table 5.7 and Table 5.8 detail the classification accuracy of pDPI, IPSVM and KISS respectively for the considered data sets. Columns report the percentages of flows (%f) and bytes (%b) classified with respect to the target classes while labels on the rows correspond to the ground truth. The percentages in bold correspond to the true positives and the true negatives correctly associated by the classifiers.

The column labeled as "Unknown" refers to traffic that does not match any signature for the pDPI, or the traffic whose statistical fingerprint is different from the target classes of IPSVM and KISS. The pDPI has one more column labeled as "Other", which reports the traffic that has been classified[3] but it does not belong to the three P2P-TV protocols under examination.

Consider Table 5.6. The accuracy of pDPI in case of Napa-Wine traces was high for PPLive and TVants, while the result with SopCast is less satisfying. Further analysis revealed that this traffic was mostly associated to Skype, suggesting that a refinement of both signatures is needed. Results are different when considering the VMware traces, which show a dramatic decline in the accuracy for PPLive. The reason is that the two data sets have been generated using two different versions of the PPLive client and the protocol resulted so different that the precision of the pDPI signature was compromised. Interesting, this effect was not noticeable in case of TVants traffic, although we used two different versions of the TVants client as well. This underline the need to continuously update of the matching rules exploited by DPI classifiers and the complexity of such process since is based on reverse engineering and manual inspection of the applications' traffic.

[3]The pDPI classifier used the June 2009 version of the NetPDL protocol database that included 72 application-level protocols (39 over TCP, 25 over UDP and 8 that operate with both TCP and UDP).

		pDPI									
		PPLive		TVants		SopCast		Other		Unknown	
		%f	%b	%f	%b	%f	%b	%f	%b	%f	%b
VMware	PPLive	**12.8**	**0.3**	-	-	-	-	9.1	98.2	78.1	1.5
	TVants	-	-	**100**	**100**	-	-	-	-	-	-
	SopCast	-	-	-	-	**91.7**	**96.0**	4.5	3.9	3.8	0.1
	Emule	-	-	-	-	-	-	**63.0**	**62.3**	**37.0**	**37.7**
	Skype	-	-	-	-	-	-	**98.9**	**97.8**	**1.1**	**2.2**
	BitTorrent	-	-	-	-	-	-	**99.7**	**99.8**	**0.3**	**0.2**
napa-IT	PPLive	**99.6**	**100**	-	-	-	-	0.4	-	-	-
	TVants	-	-	**94.1**	**100**	-	-	5.7	-	0.2	-
	SopCast	-	-	-	-	**85.4**	**90.7**	11.2	9.3	3.5	-
napa-PL	PPLive	**100**	**100**	-	-	-	-	-	-	-	-
	TVants	-	-	**100**	**100**	-	-	-	-	-	-
	SopCast	-	-	-	-	**60.8**	**80.5**	36.5	19.4	2.7	-
operator	Others	-	-	-	-	-	-	**93.2**	**99.3**	**6.8**	**0.7**

Table 5.6. Classification accuracy for pDPI.

		IPSVM							
		PPLive		TVants		SopCast		Unknown	
		%f	%b	%f	%b	%f	%b	%f	%b
VMware	PPLive	**92.1**	**89.8**	-	-	-	-	7.8	10.2
	TVants	-	-	**98.7**	**99.7**	-	-	1.3	0.3
	SopCast	-	-	-	-	**87.9**	**96.3**	12.1	3.7
	Emule	-	-	-	0.1	-	-	**100**	**99.9**
	Skype	0.2	0.2	-	-	-	-	**99.8**	**99.7**
	BitTorrent	1.4	3.9	-	-	-	-	**98.6**	**96.1**
napa-IT	PPLive	**72.0**	**7.6**	0.1	-	0.2	0.3	27.7	92.0
	TVants	0.3	-	**95.0**	**94.2**	-	-	4.7	5.8
	SopCast	-	-	-	-	**98.1**	**99.3**	1.8	0.7
napa-PL	PPLive	**83.6**	**3.5**	0.1	0.1	-	-	16.3	96.4
	TVants	-	-	**98.9**	**96.2**	-	-	1.1	3.8
	SopCast	-	-	-	-	**99.2**	**99.9**	0.8	0.1
operator	Other	0.2	-	0.1	0.1	-	-	**99.7**	**99.9**

Table 5.7. Classification accuracy for IPSVM.

		KISS							
		PPLive		TVants		SopCast		Unknown	
		%f	%b	%f	%b	%f	%b	%f	%b
VMware	PPLive	**100**	**100**	-	-	-	-	-	-
	TVants	-	-	**100**	**100**	-	-	-	-
	SopCast	-	-	-	-	**100**	**100**	-	-
	Emule	-	-	-	-	-	-	**100**	**100**
	Skype	-	-	-	-	-	-	**100**	**100**
	BitTorrent	-	-	-	-	-	-	**100**	**100**
napa-IT	PPLive	**97.3**	**4.4**	-	-	-	-	2.7	95.6
	TVants	-	-	**98.3**	**100**	-	-	1.7	-
	SopCast	-	-	-	-	**99.8**	**100**	0.2	-
napa-PL	PPLive	**95.6**	**4.2**	-	-	-	-	4.4	95.8
	TVants	-	-	**100**	**100**	-	-	-	-
	SopCast	-	-	-	-	**100**	**100**	-	-
operator	Other	24.9	1.3	0.2	34.0	0.4	0.2	**74.5**	**64.5**

Table 5.8. Classification accuracy for KISS.

The accuracy of IPSVM reported in Table 5.7 is very high for the VMware data sets, with correct results always above 89.8% (in terms of bytes) and a limited number of False Positives. Although the training was done on the VMware data set, the accuracy is high also on the Napa-Wine traces except for the PPLive traffic; the two versions generate traffic with different statistical properties that lead to misclassify the vast majority of traffic (in bytes). Interestingly, results are better in terms of flows, which seems to suggest that at least some signalling portions of the protocol are roughly equivalent in the two clients. Finally, the very limited amount of False Positives (in all traces) suggests that the classifier is robust with respect to non-P2P-TV applications.

Finally, KISS appears to produce the best results among the classifiers under testing. It reaches 100% accuracy on the VMware data set and nearly 100% also on the NapaWine data sets except for the PPLive traffic, similarly to IPSVM (i.e. many flows, but limited amount of bytes correctly classified). However, it appears to be the worst classifier on the *operator* trace, with a non-negligible number of False Positives. This is due to a peculiar characteristic of KISS, which defines an explicit class for the *unknown* traffic. Since the training was completed on the VMware traces and the unknown class included only eMule, Skype and BitTorrent (which represent a limited subset of the protocols present in real traffic), the completeness of

this class is limited and influences the accuracy when the traffic under examination is substantially different from the training set.

5.4 Overall comparison

The three classifiers are extremely different, starting from the classification method (either exact or statistical), the different information used in the classification (from packet sizes to application-level data), and the different base units (flows and endpoints).

In particular, the "traditional" pDPI is still very effective thanks to the fact that these applications do not use encryption or obfuscation techniques. Its main problem is the process of deriving the signatures that is still manual, time-consuming and error-prone. IPSVM and KISS replace the signature database with a more convenient automatic training phase and they guarantee excellent results if the training set is appropriate.

Problems appear when the training set differs from the inspected traffic (e.g. the case of KISS on the *operator* traffic, or both KISS and IPSVM with PPLive on the NapaWine traces), but this is in principle similar to the pDPI case when signatures need to be updated because of a change in the application-layer protocol. On the other hand, statistical approaches such as IPSVM that do not use topology-dependent features such as inter-arrival times have shown that they can perform reasonably well even when training information derived on one network is then applied to a different one.

Our tests show that KISS is the best classifier with respect to UDP P2P-TV traffic. This is due both to the statistical signatures and the adoption of endpoint-based algorithm (using an approach similar to [3]), that reveals to be particularly appropriate for P2P traffic in which many short flows appear from/to the same endpoint over time. However, similarly to DPI techniques, Stochastic Packet Inspection fails in case proper encryption is used by the applications, because all the bits in the payload will assume random values. Another problem is related to the usage of a multi-class SVM classifier, which requires the unknown class to be trained with samples representative of all the possible protocols. As shown in the *operator* trace, this could originate a large amount of False Positives when the protocol mix in the unknown class changes. IPSVM proves to be effective too, although its accuracy is mined by the higher percentages of False Negatives considering per-flow classifications. On the plus side, since only packet size and transmission direction are used as features, in principle it should be applicable also to encrypted traffic.

Chapter 6

Mining unclassified traffic using automatic clustering techniques

In the last years, several traffic classification techniques have been proposed ranging from port-based, DPI or statistical analysis (see Chapter 1.2). All these classifiers share some key aspects. On the one hand a deep domain knowledge is required to correctly train and periodically update these classifiers. An example of this has been shown in Chapter 5 where different release of the same application can be identified with a different accuracy. On the other hand, the classifiers can identify only the specific applications they have been trained for; all other traffic is aggregated in a single class labeled as "unclassified". The classifiers are therefore typically tuned to identify the prominent classes but they completely miss the dynamics of the rest of the traffic. For example, they cannot identify the introduction of a new application, or changes in the users' behavior or in the applications protocols.

Classification can happen at different degrees of granularity: packet, flow, or endpoint, with significant differences on the number of objects to be considered. However, when mining the subset of unclassified traffic, the number of objects to be analyzed is still large even when considering higher aggregation levels. For instance, for moderate traffic aggregates, the even small fraction of unclassified traffic is typically built by thousands of endpoints, each aggregating tens of flows made of hundreds of packets. How to practically reduce the number of unknown objects to analyze is therefore a key problem.

In this chapter, we focus our attention on the inspection of the unclassified traffic. Exploiting the KISS framework which is able to generate very representative application protocol signatures, we propose an unsupervised technique that, having no knowledge of the applications that generate the traffic, partitions a traffic aggregate into "clusters" that are distinguished based on common features, i.e., they exhibit a common treat. A simple clustering methodology based on the K-means algorithm is augmented with the capability to effectively determine the number of

traffic clusters K. The result is a simple algorithm that can reduce the number of objects to analyze to few tens, even if the total traffic amounts to several tens of megabits per seconds. By being completely automatic and unsupervised, the proposed methodology can be engineered to: i) identify new classes of traffic by exploiting the network administrator domain knowledge when inspecting a traffic cluster; ii) monitor the traffic evolution by highlighting the birth of traffic clusters corresponding to traffic of previously unobserved applications; iii) design anomalies detection techniques by observing the evolution of traffic clusters over time.

To test and validate our methodology, we consider some UDP traffic traces of which we already have a deep knowledge on, achieved through a combination of DPI and statistical techniques, as well as the results of some active experiments. We consider UDP traffic since today its importance is steadily increasing [78], and few works explicitly targeted it in the past. We apply the proposed technique to the traces and check the coherence of the automatic classification with our ground truth.

This chapter is organized as follow. We start recalling the mathematical formulation of the KISS signatures and we adapt it as to better operate in a linear space. Then, we present the aggregate clustering algorithm created and the results obtained from the considered data sets. Finally, we present some examples of classification of unknown traffic we were able to identify.

6.1 KISS Signatures Linearization

KISS signatures are computed over the packets directed to or originated from a given *endpoint*. They aim at measuring the randomness of the first bytes of the packet payload that are those usually carrying application header. In particular, the first 12 bytes of the packet payload are divided into groups of $b = 4$ bits, for a total of $G = 24$ groups. For each group, the statistic of the occurrence of each of the $2^b = 16$ possible values is computed over $N = 80$ packets. Then, the randomness of each group g, denoted by X_g, is measured as the Chi-Square distance of the group statistics with respect to the uniform distribution,

$$X_g = \sum_{i=0}^{2^b-1} \frac{(O_i^g - E_i)^2}{E_i} \tag{6.1}$$

where O_i^g is the observed occurrence of the value i for the g group, and $E_i = N/2^b$ is the expected occurrence for the uniform distribution.

From the mathematical formulation we can evince that X_g grows exponentially with the number of deterministic bits in the group. Since we are interesting in using

this features in a linear space, the signature are linearized using

$$b_g = \log_2 \left(\frac{X_g}{N} + 1 \right) \tag{6.2}$$

where b_g represents then the number of constant bits in group g. The vector $\{b_1,b_2,\ldots,b_G\}$ represents the KISS signature used in the rest of this work.

6.2 Clustering Methodology

KISS signatures can be used to map the traffic generated by applications into points in an hyper-space. To partition the space into pure clusters where points are generated by the same application, we leverage on the *K-means* algorithm, a classic unsupervised technique [4]. Given a set of K "centroids", the K-means algorithm iterates over two steps: it first assigns each point to the closest centroid, defining a cluster; then, each cluster centroid is re-computed as the arithmetic mean among all points of the cluster. The algorithm ends either after a predefined number of iterations or if centroids do not change at a given iteration. At the beginning, centroids are randomly picked.

The major drawback of K-means is that it assumes the a-priori knowledge of the number K of clusters one is interested in. The proposed algorithm tries to overcome this limitation using an agglomerate approach. We start by decomposing the hyper-space in a large number of clusters, K_0. Then, we incrementally merge the two closest clusters until one cluster only remains. A similar technique was successfully applied to the network measurement context in [7]. The pseudo-code of the algorithm is:

```
K = K0
centroids, labels = K-means(K, data)
while (K > 1)
    c1, c2 = closest_centroids(centroids)
    centroids = merge_centroids(centroids, c1, c2)
    labels = redo_labeling(data, centroids)
    K = K - 1
```

We start running K-means with $K_0 = 100$ randomly chosen centroids, i.e., we force the partitioning of the hyper-space in a large number of small clusters that are extremely pure. The algorithm then iterates merging at each step the two closest clusters: at step K, the algorithm looks for the two closest centroids $c1$,$c2$, it merges them into a new centroid positioned at the geometric barycenter of $c1$ and $c2$; then points are reassigned to the new set of $K-1$ centroids. The algorithm continue the aggregation until 2 clusters only remain.

The rationale behind the algorithm is that two centroids which are very close are likely to be associated to the same final cluster. By monitoring the value of the closest distance between centroids at each iteration step, and using this as an *indicator function*, it is possible to decide the optimal value of K, namely K_c, to stop at.

In our scenario, K_c represents the estimated number of protocols that are present in the data set. Let the smallest distance between centroids be defined as

$$\gamma_K = (d_K - d_{K-1})^2 \tag{6.3}$$

where d_K is the Euclidean distance between the two closest centroids at step K of the algorithm. γ_K defines our indicator function. Since the distance between points (and clusters) that correspond to the same protocol is expected to be smaller than the distance between points that correspond to traffic generated by different applications, large values of γ_K suggest that the algorithm is artificially enforcing the merging of two clusters that are quite different from each other.

Notice that only a single run of K-means is executed at the beginning to obtain the initial set of clusters. At each iteration, the algorithm works only on the centroids, and this has two main benefits. First, we can better control the modification on the space due to aggregation. In fact, the K-means algorithm is subject to "oscillation effect", i.e., small modifications in the centroid position could lead to large transformations in the cluster geometry. By using centroids only we avoid to re-assign samples to centroids, so that the quality of the initial clustering is better preserved.

In addition, by acting only on the centroids we reduce the computational cost by several order of magnitudes, we handle $O(K_0)$ centroids instead of $O(N)$ samples ($N >> K_0$). Moreover, the K-means complexity depends on the maximum number of iterations I (which in our case we set to 100), so that its complexity is $O(I \cdot N)$. In our experiments on an AMD Athlon-64 X2 Dual Core Processor 4200+, we elaborated several thousands of points present in a 15 minute long traffic traces in less than 3 minutes, the largest majority of the time being devoted to the initial K-Means run. Given that the code used can be further optimized, the result is promising and suggests that the algorithm might be applied to real-time monitoring.

Finally, K-means is known to suffer from the choice of the initial centroid position. Usually initial centroids are randomly chosen so that different starting conditions can lead to different clustering. In our scenario, since we select a large number K_0 of clusters, the bias introduced by the selection of the initial centroids is minimal. We performed some tests by running the algorithm with different initial random seeds and the results show that there is practically no influence on the initial choice.

ISP-Trace

Protocol	#flows $\times 10^3$ (%)		Mbytes (%)		#endp. $\times 10^3$ (%)		#sign. $\times 10^3$(%)	
BitTorrent	217	(3.39)	40	(0.19)	34	(4.14)	22	(0.33)
DNS	260	(4.05)	185	(0.88)	153	(18.79)	31	(0.47)
eMule	5200	(80.96)	936	(4.43)	476	(58.56)	61	(0.91)
RTCP	8	(0.13)	46	(0.22)	6	(0.73)	25	(0.38)
RTP	9	(0.14)	18244	(86.26)	7	(0.86)	6222	(92.14)
Unclassified	728	(11.34)	1698	(8.03)	137	(16.92)	390	(5.78)
tot	6422	(100.00)	21149	(100.00)	813	(100.00)	6751	(100.00)

P2PTV-Trace

Protocol	#flows $\times 10^3$ (%)		Mbytes (%)		#endp. $\times 10^3$ (%)		#sign. $\times 10^3$(%)	
PPLive	27	(78.52)	1585	(32.96)	184	(38.90)	23	(28.30)
SopCast	5	(14.87)	2282	(47.43)	176	(37.21)	48	(57.46)
TVants	2	(6.61)	943	(19.61)	113	(23.89)	12	(14.24)
tot	34359	(100.00)	4810	(100.00)	473	(100.00)	83	(100.00)

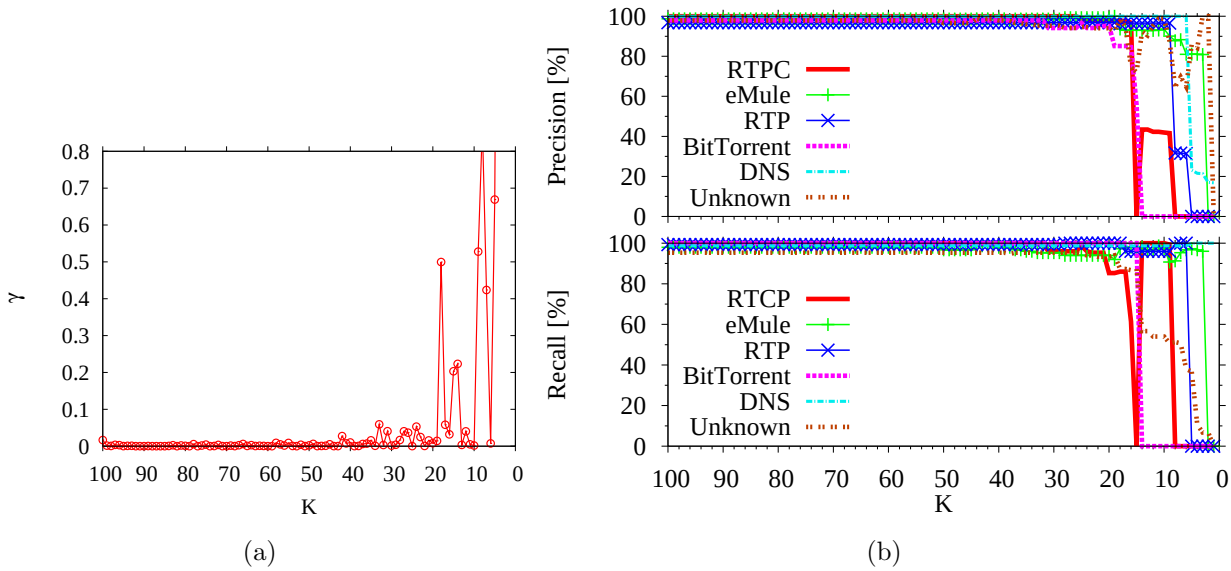
Table 6.1. Data sets descriptions.

6.3 Data sets

The results presented in this paper refer to data sets extracted from two traces, called *ISP-Trace* and *P2PTV-Trace* described in Table 6.1.

ISP-Trace: this trace corresponds to real traffic collected from the network of an Italian large ISP which offers converged services, in which data, native VoIP, and IPTV share a single broadband connection. This data set is representative of a very heterogeneous scenario, in which users are free to use the network without any restriction. It therefore is a very challenging scenario for traffic classification. In this paper we present results considering a data set obtained monitoring a PoP for 24 hours in October 2007, during which about 21GB of UDP traffic and 813,000 endpoints were monitored. Some *known* protocols (BitTorrent, eMule, RTP, RTCP and DNS) have been extracted from the aggregated trace using Tstat [24], a traffic classifier that combines a number of DPI mechanisms with statistical techniques. The classification has been manually cross-checked to have a high confidence in the ground truth. These protocols account for more than 90% of the total volume, as shown in Tab. 5.2. The remaining 10% of traffic has been labeled as "unclassified".

P2PTV-Trace: this trace was collected during ad-hoc experiments that were organized to observe the performance of popular P2P-TV applications, namely



(a) (b)

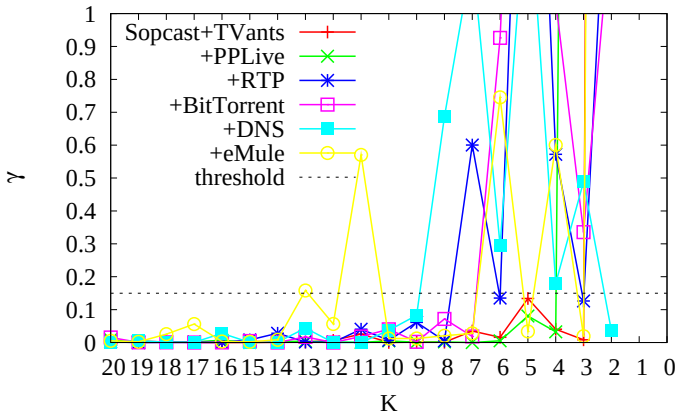
Figure 6.1. Evolution of the clustering algorithm: the indicator function (a) and classification accuracy in terms of precision and recall (b).

PPLive, SopCast and TVants. The resulting data set [13] consists of packet level traces collected from more than 45 PCs running P2P-TV applications in 5 different countries, and it is representative of a wide range of different scenarios. Being the result of active experiments, the trace contains only a single protocol at a time and we have a perfect knowledge about it.

The data sets extracted from the two traces are disjoint. In fact, there is no P2P-TV traffic in the ISP-Trace. When needed, we can artificially "inject" P2P-TV traffic from the P2PTV-Trace into the ISP-Trace as to increase the number of known protocols when assessing the performance of the clustering algorithm.

6.4 Evaluation of the agglomerate clustering approach

Figure 6.1(a) shows the evolution of the indicator function during the application of the algorithm to ISP-Trace considering a 10 minute long trace. The minimum distance between clusters is very small for values of $K > 20$, suggesting that the algorithm is merging clusters whose centroids are very close. Instead, for $K \leq 20$, the algorithm starts merging cluster centroids which are quite far from each other, suggesting an improper and artificial merging.



Classes	K_c	Precision	Recall
2	3	100	100
3	4	100	100
4	8	100	100
5	7	99.24	99.2
6	9	98.13	97.88
7	14	98.4	98.18

Figure 6.2. Example of indicator function for traffic aggregates with progressively increasing number of classes.

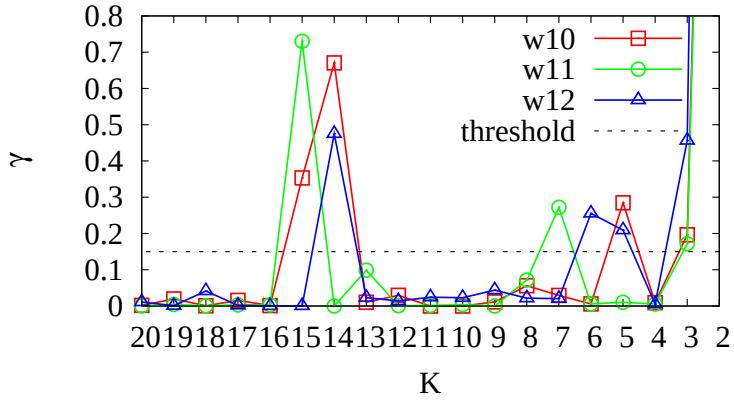
To confirm this intuition, the homogeneity of each cluster is evaluated against the endpoint classification obtained by Tstat (our ground truth). Figure 6.1(b) reports the precision (top) and recall (bottom) performance indexes, defined as

$$Precision = \frac{true\ pos}{true\ pos + false\ pos} \qquad Recall = \frac{true\ pos}{true\ pos + false\ neg} \tag{6.4}$$

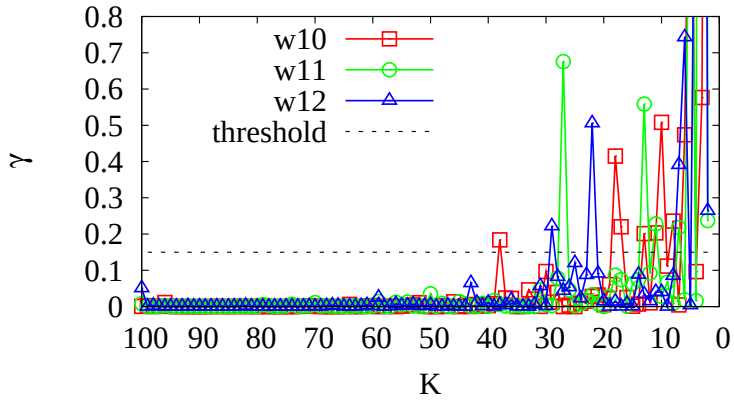
for different values of K. Precision is a measure of exactness or fidelity, whereas recall is a measure of completeness; these two measures complement each other (see Chapter 1.3).

Consider Figure 6.1(b); two observations hold. First, for $K > 20$, the fidelity and completeness of the identified clusters is very high, proving that the KISS signatures accurately represent different protocols, and that traffic generated by different applications can be easily clustered. Second, the abrupt decrease of both precision and recall observed in Figure 6.1(b) for $K \leq 20$ confirms that some clusters corresponding to different protocols are artificially merged, causing the formation of impure clusters.

To further assess the goodness of the approach, we inspect the behavior of the indicator function considering data sets in which we progressively add traffic of various applications. We start by considering a data set containing only SopCast and TVants traffic; we then add, in sequence, the traffic of PPLive, RTP, BitTorrent, DNS and eMule to the data set. For each traffic mix we run our algorithm. The results are reported in Figure 6.2 for $K \leq 20$, only. The figure shows that the indicator function abruptly increases for values of K that are strongly related with the number of traffic classes. A simple threshold mechanism on the indicator function can be adopted to automatically detect the value K_c. As an example, the figure reports a threshold of 0.15 that resulted very effective in our tests.



(a) without unclassified



(b) with unclassified

Figure 6.3. Evolution of the clustering algorithm over different time windows of the ISP-Trace.

The Table on the right of Figure 6.2 reports the suggested number of clusters K_c obtained with the threshold $\gamma = 0.15$, the corresponding Recall and Precision are also indicated. Results confirm that the value of K_c increases with the number of traffic classes. The resulting Precision and Recall are extremely high, and a marginal decrease is observed only when considering more than 5 protocols. This is due to BitTorrent traffic which is sometimes confused with TVants traffic whose KISS signatures result similar. Nevertheless, the performance are very good.

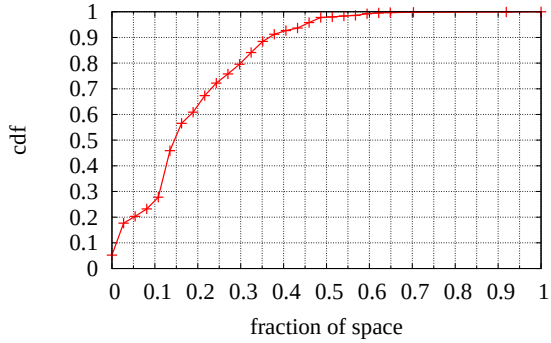
Interestingly, the number of identified clusters is larger than the actual number of applications. This is due to single applications using multiple protocols with different formats, e.g., signaling is different respect to data messages. The KISS signatures are therefore different, and the clustering algorithm correctly identifies separate clusters.

Finally, we repeat the experiment considering other different 10-min-long traces extracted from the ISP-Trace. The goal is to investigate if the indicator function always correctly suggests the number of cluster to use. Figure 6.3(a) reports the indication functions obtained for the three windows considering the aggregation of the 7 considered traffic classes. No unclassified traffic is present. We can see that the suggested K_c is consistent among all the experiments. Instead, Figure 6.3(b) reports the indicator functions when unclassified traffic is present too. In this case, since different traffic mixes are present during different periods of time, higher noise is present with respect to the previous case and different values K_c are selected in different windows. In conclusion, the indicator function suggests an optimal number of clusters which changes depending on the actual traffic mix. Thus, a conservative large number of clusters is preferable, especially when considering different time windows. Moreover, note that in Figure 6.3(b) the number of suggested clusters is never higher than 40.

6.4.1 Comparison with other clustering techniques

The automated selection of the optimal number of clusters is not new in literature. Several score indexes have been proposed to precisely correlate the goodness of the clustering with the number of used clusters. Examples of these indexes are: the Bayesian Information Criterion (BIC) adopted by the XMeans algorithm [4] and the Normalized Mutual Information (NMI) [6]. In this work, we are interested in investigating the automated approaches which do not require the a-priori knowledge of the points' labels (that is instead required by the NMI). We evaluated the performance of both XMeans and NMI; in addition, we considered also the DBScan algorithm. XMeans shows similar performance as our algorithm in terms of Recall and Accuracy. However, the number of identified clusters is typically much larger than the one obtained by our algorithm. For example, XMeans accuracy is higher than 95%, but at least 10 more clusters are identified, i.e., 50% more than with our proposal. Considering NMI, the accuracy is lower than 95% when 25 clusters are used, as suggested by the NMI technique. With 40 clusters, performance of the NMI-based method is similar to the one of our algorithm. Finally, DBScan performed poorly achieving only 85% of accuracy with the best parameter setting.

Notice also that all previous algorithms are computationally more expensive than our proposal. In conclusion, the proposed algorithm is completely automated, does not require any knowledge of the points labels and seems a good trade-off among clustering accuracy, number of clusters and complexity.



Class	#cluster	#small	#not-dense
BitTorrent	1	1	-
RTP	1	1	-
DNS	3	1	-
RTCP	7	2	4
eMule	10	6	7
Unclassified	22	20	18

#small: n.o. clusters with radius ≤ 0.2
#not-dense: n.o. clusters with ≤ 10 points

Figure 6.4. Cumulative distribution function of the distance between points and centroid in each cluster (plot on the left) and a few additional data (table on the right), obtained running the algorithm on ISP-Trace with $K = 40$.

6.4.2 Clusters distances

In this section, we investigate the geometry of the clusters of points identified by our algorithm. The results presented in this section are obtained using $K = 40$ (a conservative large value) and refer to a single time window of ISP-Trace. For the other time windows, not shown here for the sake of brevity, we obtained similar results.

We start by considering the size of each cluster. Figure 6.4 shows the cumulative distribution function of the normalized Euclidean distance between each point and its centroid. As we can see, half of the points in the data set are very close to theirs centroid, with a distance smaller than 20% of the cluster space size. The table on the right of Figure 6.4 reports some statistics about the clusters geometry according to the DPI classification. In particular, the second column reports the number of clusters identified for each application, the third column reports the number of *small* clusters, i.e., clusters with a radius smaller that 0.2, and the last column gives the number of *not-dense* clusters, i.e., clusters with less than 10 samples. BitTorrent and RTP are mapped into a single cluster, while the "unclassified" is composed of a set of small, often not-dense, clusters. Interestingly, eMule is highly partitioned too. Investigating further, we noticed that each cluster corresponds to a different protocol which eMule uses for different purposes, e.g., one protocol is used to exchange messages with the server, another one is used to exchange traffic with peers.

To better understand the possible overlapping between the clusters, Figure 6.5 reports the distance between pairs of centroids. The distance has been mapped onto a gray scale in which the darker the color, the nearer the centroids. The image is symmetrical with respect to the main diagonal, where all points have a distance of 0 by definition. Clusters are ordered based on their type of traffic so that clusters

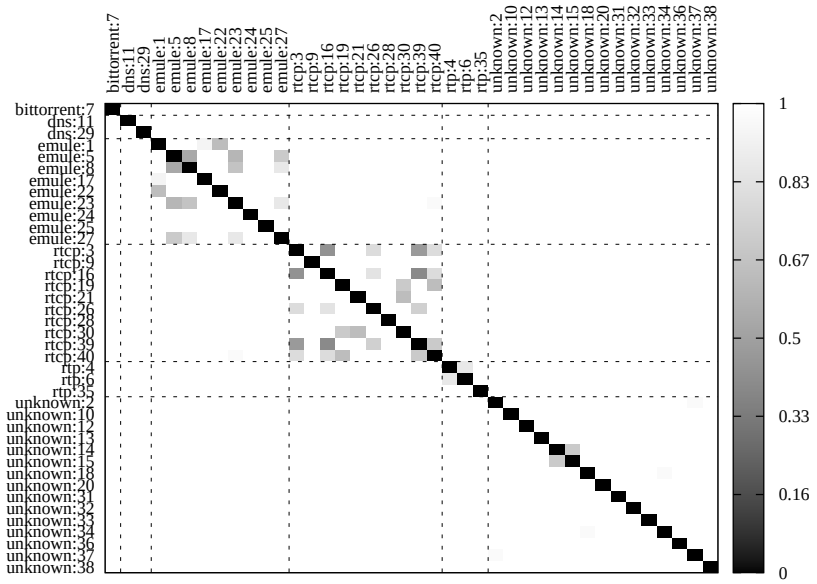


Figure 6.5. Distance between centroids of different clusters, for 40 clusters obtained by running the algorithm on IPS-Trace with $K = 40$.

referring to the same application are nearby; dashed lines are used to delimit the applications. The only blocks which include nearby clusters are related to the same application.

In conclusion, we can say that the KISS signatures map different protocols in different compact clusters of the hyper-space. The geometry of the clouds is strictly related to the characteristics of the application, but the signatures are naturally clustered in pure areas which do not overlap.

6.5 Mining the unclassified traffic

In this section we show how the proposed technique can be used to monitor the traffic evolution in time and detect the presence or absence of traffic in different periods. To do so, we measure the modifications of the clouds obtained by running the algorithm over consecutive time windows. We consider 1 hours of traffic divided into six 10-min-long traces and for each trace we run our algorithm using $K = 40$, as previously described. The centroids obtained for each time window are then compared with centroids identified in the previous time window. Each centroid is associated to the closest cluster in the previous set according to their geometric distance. This allows to detect changes between the current and the previous cluster

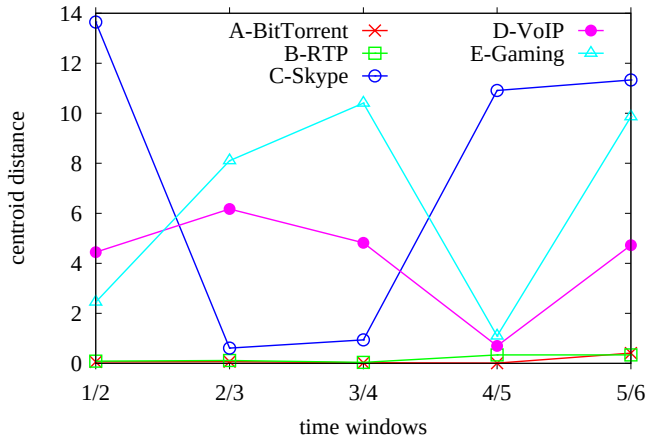


Figure 6.6. Example of evolution of the centroids position.

placement.

Figure 6.6 reports some interesting examples; it shows the distance of some selected centroids in consecutive time windows. For example, the position of centroid A and centroid B is practically the same over time. Verifying the corresponding clusters, we found out that samples of cluster A and cluster B are associated to BitTorrent and RTP, respectively; since in the traffic traces those applications are always present, the corresponding centroids are always present and more or less in the same position.

Consider now the case of the cluster with centroid C. The minimum distance among the centroid C in the first and the second time window is very high, suggesting that in the second time window C is associated to a cluster of traffic that was not present during the previous time window. When comparing centroid C to its closest centroid at time window 3, we see that it moved very little. Similarly, considering time windows 3 and 4, centroid C is still referring to the same cloud of samples. Only in time window 5, the centroid C seems to disappear, since the closest centroid is very far from its position during time window 4. This suggests that some new traffic appears at the 2-nd time window, it is present during the 3-rd and 4-th time window, when it disappears again. Investigating further, we discovered that the traffic was generated by a Skype call that lasted for that period of traffic. Centroid C then refers to Skype Voice protocol.

Similar conclusions can be drawn following centroid D and centroid E evolution. Comparing their position during the 4-th and the 5-th window, we can observe that they moved little, i.e., they refer to the same cluster. Manual inspection revealed that the traffic of cluster-D corresponds to STUN protocol - Simple Traversal of User Datagram Protocol that was initiated by some P2P client that was alive in

time window 4 and 5. Centroid E refers, instead, to traffic between hosts that used port 16567. This latter is composed by both short packets and much bigger packets, which might be related to Battlefield2 protocol.

Beside these examples, the methodology identified other sets of clusters and centroids which were always placed in the same zone across consecutive time windows. Some of these clusters were due to long-lived, single connections carrying many bytes, while others contained P2P-like flows, i.e. endpoints exchanging limited amount of data with an large number of hosts. Unfortunately, because of the limited amount of available payload, we are not able to further identify the application that generated these flows.

These examples show how we could successfully employ our technique to get insights into the unclassified traffic that Tstat DPI and behavioral classifiers cannot identify. In terms of traffic volumes, we could correctly identify and clusterize more than the 40% of unclassified traffic.

6.6 Conclusion

In this chapter we presented a clustering methodology to partition a traffic aggregate in classes according to the generating application. Using statistical signatures as those of KISS, one of our classifiers, the methodology (that is completely unsupervised) is based on the K-Mean clustering algorithm enhanced through a mechanism to detect the optimal number of clusters.

Results show that the traffic partitions are very accurate and confirm that the statistical signatures are effective in capturing the differences among application protocols. Moreover, our results prove that the methodology can be effectively used in different contexts. First of all, it is helpful to mine the unclassified traffic, i.e. the traffic that traditional DPI or a behavioral classifiers cannot recognize. Indeed, it helped us revealing 40% of the traffic we could not classify with our classifiers. Second, the algorithm can reveal the born of new applications, as well as the changes of existing ones.

Part II

Dissecting YouTube Video Streaming System

Chapter 7

YouTube CDN overview

Created in 2005 and bought by Google in November 2006, YouTube is the most popular and bandwidth intensive service of today's Internet: it accounts for 20-35% of the Internet traffic [27, 49, 45] with 35 hours of videos uploaded every minute and more than 700 billion playbacks in 2010 [75, 77]. With such a high popularity, it presents a challenge both for the system itself and for the Internet Service Providers (ISP) that need to offer good quality of service for video streaming demands. Therefore the YouTube phenomenon attracted the interest of the research community, with several works [28, 79, 11, 12, 66] focusing on either video characterization, infrastructure, or user behavior. A recent notable work [1] has greatly contributed to the understanding of the YouTube Content Distribution Network (CDN) through an in-depth analysis of traffic traces of a tier-1 Internet Service Provider (ISP). However, much of this analysis has focused on the architecture prior to the acquisition of YouTube by Google Inc. It is unclear to what extent these observations continue to hold today.

A second recent change in the way people access the Internet is due to the exploding popularity of mobile devices. Smartphones and Internet tablets are today commonly used both at home and at public places, and the phenomenon is still growing in popularity. Recent estimates forecast that within a few years mobile devices will be the users' preferred choice for accessing the Internet [52] while according to [49, 70] multimedia content represents already a big share of the mobile traffic, with YouTube as the main contributor. Still, mobile operators are struggling with the intrinsically limited capacity of mobile access technologies.

The mix of the two phenomena has serious implications for both content providers and ISPs. Indeed, while YouTube is already commonly accessible on mobile devices from 3G/4G networks, the video encoding rate (and quality) is, by design, much more limited than the one offered to PCs. At the same time, mobile ISPs adopt tariff plans with the explicit goal to limit the amount of traffic a device can consume, a trend that is becoming popular among wired ISPs as well.

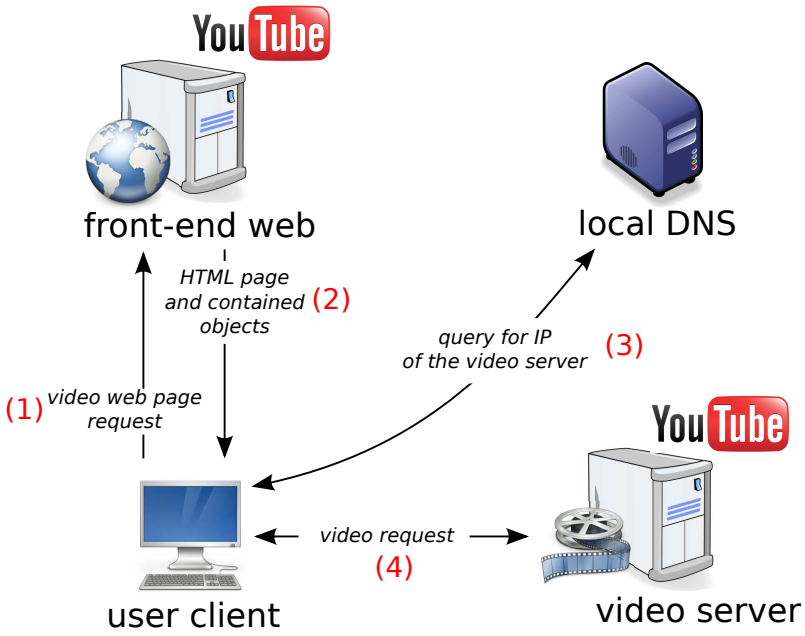


Figure 7.1. High level sequence of steps to retrieve a video.

In the remaining of this chapter, we give an overview of the YouTube CDN presenting the elements that compose the system and the adopted video delivery mechanisms. The resulting scenario is complex, with different entities interacting while policies are used to control both the communications and the video delivery.

7.1 YouTube CDN elements

YouTube is a web-based service so users access to the videos through a web browser which either has the Adobe Flash player installed or is HTML5 compliant. The client interacts with three elements which compose the system as reported in Figure 7.1:

- **Front-end web**, serving the YouTube portal[1];
- **Video server**, caching and serving videos;
- **DNS**, translating YouTube objects' name into IP addresses with respect to specific policies;

The videos are accessible from the YouTube portal, which is a common web site where the users browse the videos organized in categories and *channels*[2]. Each

[1] www.youtube.com, m.youtube.com
[2] In YouTube terminology, a channel corresponds to all the videos uploaded by a user.

video has a *video web page* collecting a rich set of information, e.g., video rating, number of views, comments, list of related videos, etc. When the user selects a video, he is asking to the system a specific video web page performing the HTTP query `http://www.youtube.com/watch?v=videoID` where `videoID` corresponds to a 11 character long string uniquely identifying a video (step 1).

The web front-end contacted returns a HTML page containing several objects (e.g., images, javascript files, etc.) which have to be retrieved in order to view the web page. Among these object there is also the embedded Adobe Flash player that takes care of the download and playback of the video (step 2). The name of the video server that will provide the video is among the parameters provided for the Flash object and it is encoded using a static URL. Then, the video server name is resolved to an IP address by the client via a DNS query to the local DNS server (step 3). Finally, the client will query the video server to get the actual video data (step 4).

We can group the four steps in two phases: the interactions related with the web front-end (step 1 and 2) are used for the *content look-up*; the interactions related with the video server (step 3 and 4) are used for the *content download and playback*. In fact, it is interesting to notice that, even if is the web-server that specify the video server name while generating the video web page for the client, it is the DNS that translate the name to an IP. This resolution is not arbitrary and it is exploited by YouTube to route clients to appropriate servers according to some internal policies. Moreover, suppose that the video server contacted cannot provide the requested video. The client then has to retrieve it from another server and a *redirection* occurs.

These hidden mechanisms represent the core of the system and have a key role in assessing the download performance. Consider for example the case in which the DNS forces to download a video from a "far" location. It is obvious than that the download is more subject to sudden congestions and losses which can affect the Quality of Experince (QoE) perceived by the user. Similarly, if the video server contacted do not have the requested video, the user has to look for it somewhere and the *startup latency* increases, i.e., the user have to wait longer before the playback can start.

7.2 Users devices

YouTube can be accessed from a wide range of devices, each with different capabilities and hardware constraints. Depending on the client device, two mechanisms are used to retrieve the video content:

PC-player: the client is a regular PC running either a web browser with the

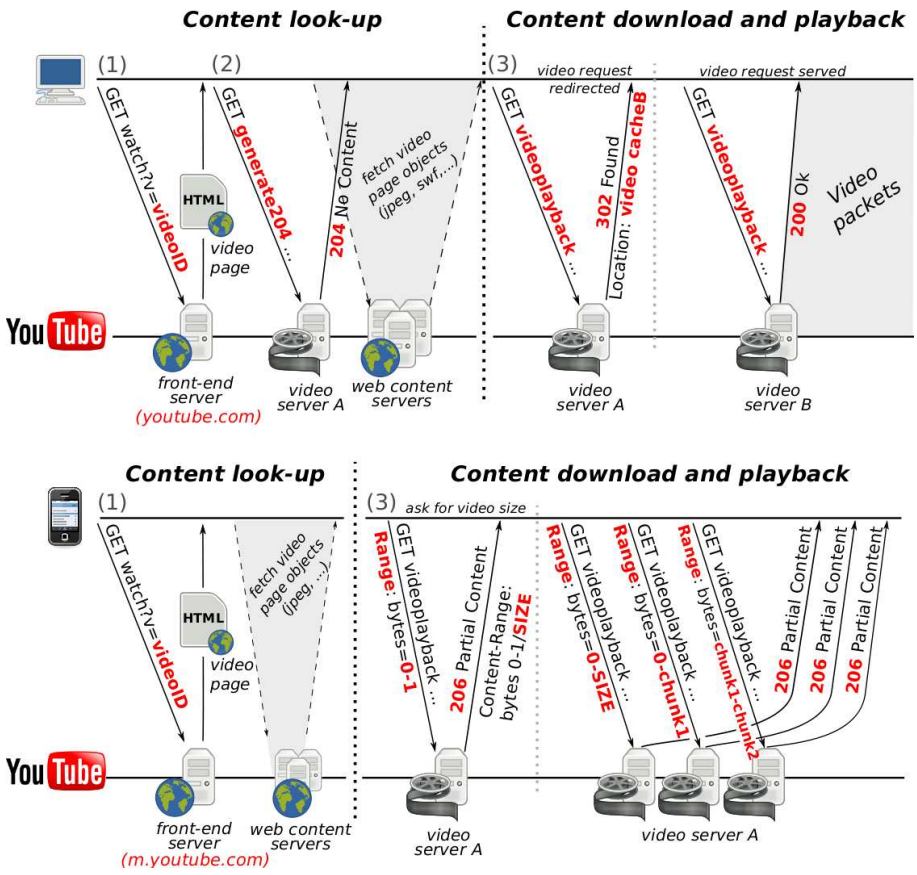


Figure 7.2. YouTube video download mechanisms. Example of possible evolution when accessing to *youtube.com* from a PC (top) and *m.youtube.com* from a smartphone (bottom).

Adobe Flash plugin or HTML5 compliant browser[3]. We performed experiments with several browsers and operating systems and found no difference in the traffic they exchange during the second phase. Hence, we will refer to them as PC-player without further distinction[4].

Mobile-player: the client is a smartphone, an Internet tablet or a set-top-box which uses a custom application[5]. Also in this case we tested different combination of devices running both Apple iOS, Google Android and other operating systems. While several differences are found when considering the first phase,

[3]http://www.youtube.com/html5

[4]Notebooks and netbooks using regular browsers belong to the PC-player category.

[5]Even if set-top-boxes and TV appliances are hardly mobile, they use the same access mechanism as smartphones, so we consider them in the Mobile-player category.

they all behave similarly in the second phase [62]. Therefore, we will refer to them as Mobile-player.

7.2.1 PC-player

Let us consider a client accessing the `www.youtube.com` web site from a regular PC using a browser as shown in Figure 7.2 (top). We can split the interaction between the browser and the YouTube servers in three steps: (1) video web page retrieval, (2) video prefetch and (3) video download.

During (1), the client downloads the web page describing the video. The HTML document contains a combination of text and other "objects" (e.g., the Adobe Flash player) that the browser needs to fetch to properly display the page. Among the different objects, a Javascript function triggers a `generate204` request sent to the video server that is supposed to serve the video. This starts the video prefetch (2), which has two main goals: first, it forces the client to perform the DNS resolution of the video server hostname. Second, it forces the client to open a TCP connection toward the video server. Both help to speed-up the video download phase. In addition, the `generate204` request has exactly the same format and options of the real video download request, so that the video server is eventually warned that a client will possibly download that video very soon. Note that the video server replies with a `'204 No Content'` response, as implied by the command name, and no video content is downloaded so far.

At this point, the browser handles the control to the Flash player which will manage the actual video download (3). The player sends a HTTP `videoplayback` request to get the video. Note that the same TCP connection previously opened during (2) can be used if HTTP persistent capability is supported between the browser and the Flash plugin. Because of server congestion or lack of content, the server can *redirect* the client to other servers. In this case, the video server replies with a HTTP `'302 Found'` response which specifies the hostname of another video server to contact. The player then resolves the hostname, and sends a new `videoplayback` request. This process can repeat until a valid video server is found. The final video server of the chain replies with the usual HTTP `'200 OK'` response, which initiates the stream of video data to the client.

We highlight that the `generate204` request is a specific optimization that is found only when accessing a video through `www.youtube.com`. YouTube videos embedded in regular HTML pages do not exploit this, so the player have to resolve the video server hostname yet before sending the first `videoplayback` request.

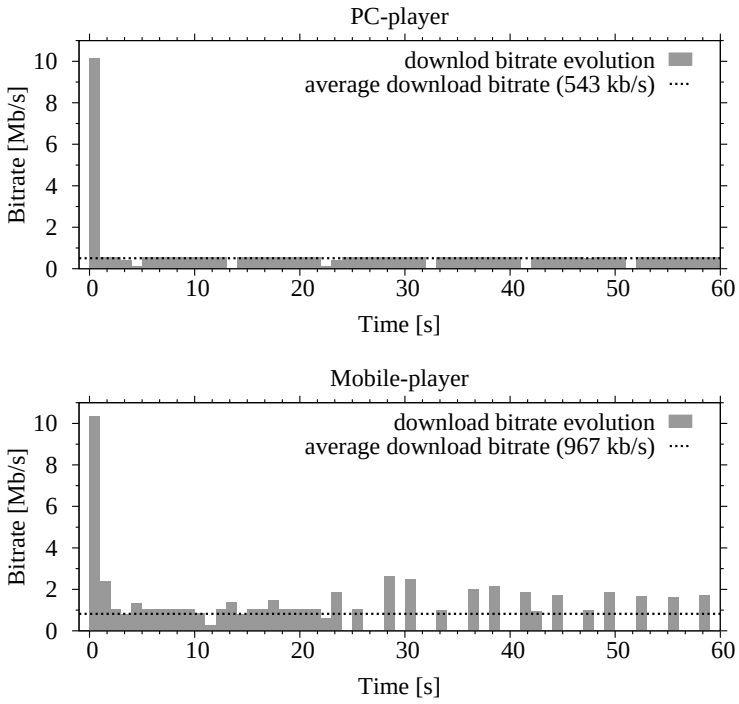


Figure 7.3. Evolution over time of the download bitrate for a video encoded at 540 kb/s.

7.2.2 Mobile-player

Mobile devices use a different protocol as shown in bottom part of Figure 7.2. First, no prefetch message is sent in the content look-up phase. Second, differently from the PC-player case, the video content is downloaded in "chunks", each one requested in a separate TCP connection, using the HTTP `Range` header field to specify the requested portion of the video. The video server then replies with a `'206 Partial Content'` response.

This mechanism is possibly the result of a design choice that tries to cope with the tighter constraints in terms of storage availability for mobile devices. In fact, the mobile devices can hardly buffer the entire video so the player progressively requests portions according to the evolution of the playback.

7.2.3 Example of video download

To illustrate the differences in the video delivery mechanisms, Figure 7.3 shows the bitrate evolution obtained downloading the same video from a PC (top) and a mobile device (bottom). In both cases, the server starts sending an initial burst

of data at a very fast rate to quickly fill the play-out buffer at the player. This mechanism is conventionally called "fast-start". The server then starts shaping the rate as observed in [2]. Note that this is a server-based shaping mechanism in which the client has no role (neither application layer nor TCP flow control messages are sent). For PC-player, after the initial burst, the download proceeds within the same single TCP connection, whose throughput is practically equal to the average video encoding rate (dotted line). Note that the average download rate is computed discarding the initial burst.

For Mobile-player instead, the bitrate evolution is more bursty. This is a consequence of leveraging different TCP connections to download chunks of video. Indeed, from second 23 and on, the mobile terminal aborts the ongoing TCP connection, and starts requesting chunks of video on separate TCP connections. They last about 1 second and are separated in time by about 2 seconds of silence. Since a new TCP connection is used, the server enters the "fast-start" phase, which is early interrupted when the client aborts the underlying TCP connection. We believe this mechanism is due to a client-side buffer management policy which abruptly interrupts the TCP connection when the play-out buffer is filled up. The client then re-starts the download when the buffer depletes below a certain threshold. This results in an inflation of TCP connections, and a possible inefficient download.

The early abortion of the TCP connection can be due to other causes as well. For example, a resolution change or a fast-forward in the video are handled by aborting the current download and starting a new one for both PC-player and Mobile-player. Finally, the initial control messages possibly sent on separate TCP connections are also fundamentals to capture the dynamics of the download.

Chapter 8

Methodology and data collections

8.1 Introduction

In the previous chapter we have seen that YouTube is a complex system, with elements interacting and controlled by hidden policies. To study such a system, we can exploit *active* and *passive* approaches. In the first case, we create ad-hoc experiments querying the system as to collect specific information. For example, exploiting the YouTube API[74], we can retrieve information on the content served (e.g., video title, popularity, ranking, etc.). In the second case instead, we monitor the activity real network users accessing YouTube. In this case, an inspection tool is required as to i) identify the YouTube traffic and ii) extract the semantic of the user's interactions with the system.

In the last years, several works have been published, focusing on different aspects of the system or the users' behaviour. We can group these works in three categories:

Video content: these works have focused on characterizing various aspects of YouTube videos as well the usage patterns. On the one hand, [28] and [79] characterized video popularity, durations, size and playback bitrate, as well as usage pattern statistics such as day versus night accesses and volume of traffic. On the other hand, [11] and [12] crawled the YouTube site for an extended period of time and performed video popularity and user behavior analysis. Further, [11] compares YouTube to other video providers such as Netflix and [12] investigates social networking in YouTube videos;

Infrastructure: these works characterize the YouTube video delivery infrastructure [1, 66]. [66] shows that most YouTube videos are distributed from a single data center in the U.S. [1] shows that a few data centers in the U.S. were in charge of distributing the videos around the world.

User Behavior on Mobile Devices: More recently, there have been several works characterizing high level usage patterns of mobile devices [49, 27, 70]. [49] shows that the number of mobile devices doubled between 2009 and 2010 and that more than 80% of mobile devices traffic is HTTP, with multimedia traffic alone accounting for more than 30% of HTTP. [27] compares the content and flow characteristics of mobile devices and PCs traffic. Using a DPI tool, the authors are able to show that YouTube alone accounts for more than 35% of the Internet traffic.

Beside this strong effort, many aspects of the system are still unknown. The majority of the works refer to the "old" YouTube, i.e., before the acquisition by Google. Very little is known about the system mechanisms with respect to the different type of devices used to download the videos. Moreover, while active experiments are very useful to investigate on the properties of content hosted in the system, these information do not picture the real usage of the system. For example, when a user upload a video, this is automatically converted in different video formats [76]. The set of available resolutions can be easily retrieved interrogating the system with the YouTube API but only through passive measurements we can actually know which is the resolution used by the users.

Considering these motivations, we adopted mainly a passive approach, collecting data from the real users activity while active measurements are used only to integrate or crosscheck the results obtained from the collected data sets. Through an extensive monitoring of the users' activity, we aim at shed some light on YouTube covering different aspects of the system ranging from the internal policies (e.g., Where the video is coming from?), video properties (e.g., What is the typical size and duration of the videos?), users behaviour (e.g., Do the users watch the whole video?) and performance (e.g., How long the users wait before the playback can start?)

In the following, we present the passive monitoring tool used to collect YouTube traffic. The data set are then described, characterizing the volumes and comparing the results with respect to the different type of devices considered. Finally, we conclude introducing the concept of *video session* which will be our key tool in the following chapters to study both the architecture and the performance of YouTube.

8.2 Collection tool

To inspect the network traffic we relied on Tstat [24, 72], an Open Source packet sniffer with Deep Packet Inspection (DPI) capabilities, which implements both traffic classifiers and fine-grained flow-level statistics. Tstat identifies the application that generates TCP/UDP flows using a combination of Deep Packet Inspection (DPI) and statistical classifiers. Tstat was found to perform well in [60].

Leveraging on this, we improved Tstat so as to identify the YouTube traffic and we distinguish all possible HTTP messages that can be observed when a client downloads a YouTube video both from the portal or third party sites such as news sites or blogs where the video is "embedded". By parsing the URL of the HTTP messages, Tstat identifies PC-player and Mobile-player accesses[1] and extracts specific video information, such as the `videoID` and the video format. We also extract other video related information from the video header, such as the resolution, total duration and size of the video.

Tstat identifies not only the control messages and the videos flows but also *site flows* (search queries, download of thumbnails and others user's interactions with the YouTube website). However, for the results reported in the following chapters, we mainly focus on the first two categories.

In addition to the video properties, Tstat also collects several TCP flow-level statistics, such as the total number of packets and bytes transmitted and received, the total flow duration and the average RTT[2]. Further information on Tstat capabilities as well as the source code can be obtained from [24, 72].

8.2.1 Data sets

Using Tstat, we collected data sets corresponding to flow-level logs where each line reports a set of statistics related to each YouTube flow. Among other metrics, the source and destination IP addresses, the total number of bytes, the starting and ending time and both the `videoID` and the resolution of the video requested are available.

We collected data sets from five locations spread across three countries including Points-of Presence (PoP) in nation-wide ISPs and University campuses. In all cases, a high-end PC running Tstat was installed to analyze in real time all the packets going to and coming from all the hosts in the monitored PoPs. We performed two separate collections of one week time period, the first between September 4th and September 10th, 2010 (YT-2010) and the second one on February 25th, 2011 (YT-2011). From all vantage points we started the collection at 12:00am, local time.

Table 8.1 summarizes the characteristics for both data sets, reporting the name, the type of users and the volumes of flows and bytes collected. The last three columns instead correspond to the number of distinct IP addresses considering both YouTube servers and clients in the PoP, and the number of distinct videos requested.

The two collection are quite heterogeneous, covering different type of users, locations and technology access. In particular, we can identify

[1]URL requests from mobile devices contain `app=youtube_gdata` or `app=youtube_mobile`. We do not make any further distinction on the type of browser or mobile device used.

[2]RTT is estimated by leveraging TCP acknowledgements [24, 72]

(a) YT-2010

Name	Type	Flows	Vol. [GB]	Servers	SrcIP	Videos
US-Campus	Campus	874649	7061	1985	20443	332146
EU1-Campus	Campus	134789	580	1102	1113	40437
EU1-ADSL	Home	877443	3709	1977	8348	259627
EU1-FTTH	Home	91955	463	1081	997	30290
EU2-ADSL	Home	513403	2834	1637	6552	173280

(b) YT-2011

Name	Type	Flows	Vol.[GB]	Servers	SrcIP	Videos
US-Campus	Campus	2172250	10898	1889	20455	446870
EU1-Campus	Campus	173024	714	1275	1203	50205
EU1-ADSL	Home	740330	2615	2538	8154	189788
EU1-FTTH	Home	135907	480	1648	1136	33762
EU2-ADSL	Home	830476	3688	2043	5826	205802

Table 8.1. YouTube data sets description

ISP Networks: the data sets have been collected from nation-wide ISPs in two different European countries. EU1-ADSL and EU1-FTTH refer to data collected from two distinct PoPs within the same ISP. The two PoPs differ in the type of Internet access technology of their hosted customers. In EU1-ADSL, all customers are connected through ADSL links and in EU1-FTTH, all customers are connected through FTTH links. The EU1 ISP is the second largest provider in its country. The EU2 data set has been collected at a PoP of the largest ISP in a different country.

Campus Networks: The data sets have been collected using a methodology similar to the ISP setting. The Tstat PC is located at the edge of each campus network, and all incoming and outgoing traffic is exposed to the monitor. We collected data sets from two University campus networks, one in the U.S. and one in a European country.

Comparing the size of the data set, they are very similar. The main difference between the collections is in the Tstat capabilities to track YouTube traffic. In fact, in the older data set YT-2010, Tstat was tuned to identify only a specific subset of control messages (mainly redirection messages) and it was not able to distinguish between PC-player and Mobile-player downloads. The two collections are then used for two different purposes: YT-2010 has been used mainly to study the internal system architecture while YT-2011 to study similarity and differences with respect to different users' devices.

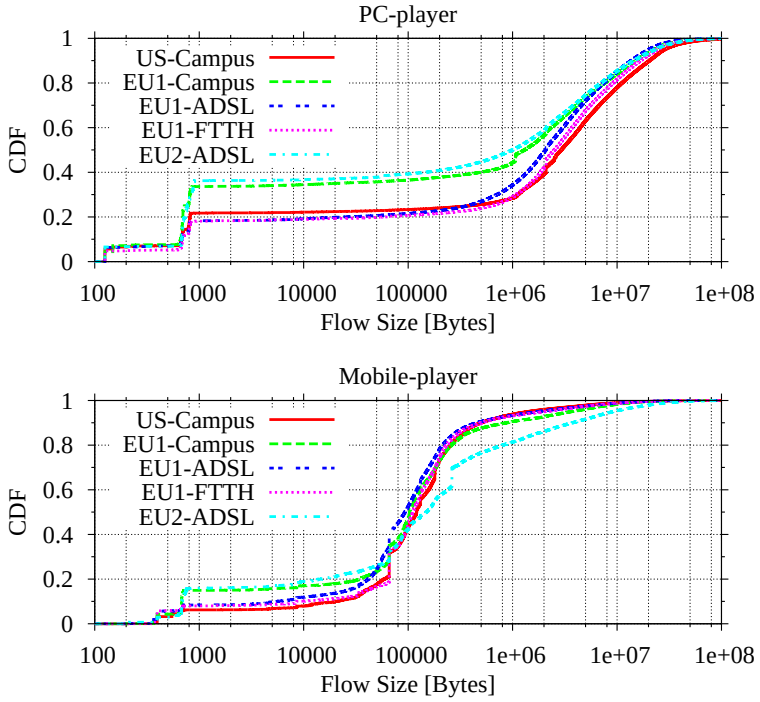


Figure 8.1. Distribution of the TCP flow size in YT-2011.

8.2.2 Video and control flows

In Chapter 7.1, we have seen that part of the interaction between the client and YouTube are related to control messages. Considering the YT-2011 data collection, Figure 8.1 reports the Cumulative Distribution Function (CDF) of TCP flow sizes, i.e., number of bytes (B) received by the client in a flow. Let us focus on the PC-player traffic (top plot). Steps in the CDF clearly show the presence of flows of typical size corresponding to specific HTTP messages: `'204 No Content'` flows (prefetching) are about 120 B long, `'302 Found'` flows (redirections) are in the [800-1000] B range, while flows containing the `200 OK` responses are typically longer than 80 kB since they contain the actual video data. Interestingly, the initial part of the distribution is different for different probes, with EU1-Campus and EU2-ADSL suffering a higher fraction of redirections (`'302 Found'`) messages. However, the tail of the distribution looks rather similar, suggesting that the size of videos downloaded in the networks monitored is similar. We will detail this better in Chapter 9.

Looking at results for Mobile-player (bottom plot), we observe that the flow size is similar across data sets, with EU1-Campus and EU2-ADSL still exhibiting higher fraction of redirection messages. However, comparing PC-player and

	PC-player				Mobile-player		
Data set	200	204	206	302	200	206	302
US-Campus	44.1	49.2	2.5	4.2	2.9	89.9	7.2
EU1-Campus	39.6	44.3	4.7	11.4	2.1	80.5	17.4
EU1-ADSL	45.5	46.2	3.6	4.7	1.4	92.4	6.2
EU1-FTTH	46.1	47.4	2.4	4.1	1.1	92.3	6.6
EU2-ADSL	40.8	40.8	4.3	14.1	14.8	66.1	19.1

`200 OK` (normal condition)
`302 Found` (redirection)
`204 No Content` (empty control message)
`206 Partial Content` (delivery of a chunk)

Table 8.2. Distribution of HTTP return code in YT-2011.

Mobile-player we observe interesting differences: (i) the absence of the prefetching phase causes the `'204 No Content'` responses, of size 120B, to disappear in Mobile-player; (ii) the abundant presence of the HTTP requests using the `Range` header field causes the flows carrying the video data to be one order of magnitude shorter than in PC-player. This is a direct artifact of the video chunking mechanisms and not a difference in the actual video duration and size (see Figure 9.1 and Figure 9.2). Interestingly, the 500B long flows are due to `'206 Partial Content'` replies to the first `videoplayback` request using the `'Range: bytes 0-1'` header which Mobile-player uses to discover the actual video size from the HTTP response field (see Sec. 7.2.2).

The differences in the methodologies are visible also considering the HTTP response codes reported in Table 8.2. For each PC-player download there is always a `204 No Content` (related to the `generate204`) and a `200 OK` (related to the video query), while for Mobile-player all the flows usually associated to a `206 Partial Content` (related to a chunk of the video). Some mobile device can also receive a `200 OK` messages [3]. Instead, the fraction of `206 Partial Content` related to PC-player is the result of an optimization of the system and occurs when the user reload the page before having completed the download. In this situation, the player have already buffered a portion of the video so it only requests the remaining part. We can also notice that the redirections are not negligible and are more frequent for mobile devices.

The effect of the chunking mechanism adopted by Mobile-player has clearly an

[3] Apple's products receives always `206 Partial Content` while other devices receive also the `200 OK`

Name	%Flows	%Bytes
US-Campus	32.5	3.5
EU1-Campus	15.6	2.8
EU1-ADSL	27.2	3.9
EU1-FTTH	42.2	6.6
EU2-ADSL	4.2	1.6

Table 8.3. Fraction of flows and bytes due to mobile terminals in YT-2011.

impact on the number of flows generated by mobile devices to download the content. Table 8.3 quantifies this by reporting the fraction of flows and bytes that are due to Mobile-player for the different data sets. We can notice that, while Mobile-player traffic is a small fraction of the total volume, it accounts for a much larger fraction of flows. This might pose performance issues on flow-based devices, like NAT boxes or full-state firewalls which keep per-flow state.

Consider now the volume of bytes. Unexpectedly, only less than 6% of YouTube traffic is due to users from mobile devices. The networks we consider offer both wired and WiFi access with large penetration of smartphones, especially in the campus networks. Therefore, one would expect that a large fraction of YouTube accesses is done from such terminals. Our measurements contrast this intuition. Moreover, some recent studies [27, 49] show that multimedia content is responsible for more than 40% of the total volume due to wireless terminals, with YouTube as the main contributor. Our results show that this traffic is little compared to the volume generated by wired networks. A possible explanation for this is the fact that wireless users in our networks still prefer to access YouTube videos from standard PC browsers, because of the better user experience compared to smartphones.

8.3 Video session definition

Tracking single flows is not enough to capture the whole evolution of a YouTube video download. As seen in Chapter 7.2, in the normal scenario each YouTube video request corresponds to a HTTP message exchanged between the Flash plugin and a video content server. If the request succeeds, then the content server starts to deliver the video inside the opened connection. It is possible however that the server may not serve the content. In such a case, it would simply *redirect* the user to another content server and close the connection. For Mobile-player instead multiple TCP connections are used, each one carrying a portion of the video. Moreover, whenever the user perform a action on the player controls, e.g., a *resolution switch*, a new TCP connection is initiated abruptly aborting the current one.

To capture such a variety of behaviors, we use the concept of *video session*, i.e.,

a mechanism to group all connections related to the download of the same content. More specifically, a video session corresponds to the set of TCP connections that

- share the same source IP address and `videoID`;
- are separated by a silence period shorter than T seconds. For instance, two connections c_1 and c_2 belong to the same session if the difference in time between the beginning of c_2 (time of the TCP SYN packet) and the end of c_1 (time of the last packet observed) is smaller than T.

The value of the threshold T is related to the type of analysis we are interested in. Small values are suited to study system interactions triggered automatically (e.g., redirections), while large values of T may also group flows generated by user interactions with the video player, (e.g., resolution switch).

Inside each session, we can further distinguish between *video flows*, i.e., long connections carrying the requested video, and *control flows*, i.e., short connections carrying signaling messages. As to distinguish the flow type we used a simple heuristic based on the HTTP return codes and the flow size as reported in Figure 8.1: flows smaller than 1000 bytes and not associated to a `200 OK` correspond to *control flows*, while the remaining corresponds to *video flows*. We have conducted manual experiments which have confirmed that flows smaller than 1000 bytes are indeed control messages. Knowledge of control flows associated with a video flow can help provide important insights for our analysis. For instance, a video flow from a user to a given server preceded closely (in time) by a control flow to another server is an indication of *redirection*. In contrast, an isolated video flow not preceded by other control flows is an indication that the request was directly served by the contacted server.

Chapter 9

Video content

9.1 Introduction

In this chapter we study the properties of the video hosted in the YouTube CDN. Some works [28, 27, 79] have already analyzed properties as the video size and duration. [11] and [12] crawled the YouTube site for an extended period of time and performed video popularity and investigate on social networking activities related to the videos. Our work anyway goes in a different direction, aiming at a deeper knowledge of the properties of the videos. Taking advantage of the YT-2011 data sets (see Chapter 8.2.1), we do not simply measure the video size, duration, average bitrate and video formats, but we consistently compare the video properties with respect to different type of users, locations, access technologies and type of device used to access to the video.

Results show that the size and duration of the videos in the system is independent from users location. More surprisingly, this is true also considering the type of device used to access to the system. As expected, the average video bitrate instead is directly related to the video resolution. However, if on the one hand 1 Mb/s is enough to cope with the most common resolution (360p), on the other hand High Definition (HD) resolutions (720p and 1080p) request up to 6 Mb/s representing then a challenge due to the high bandwidth requirements of the streaming. Finally, PC-player and Mobile-player downloads have by design two different default video formats.

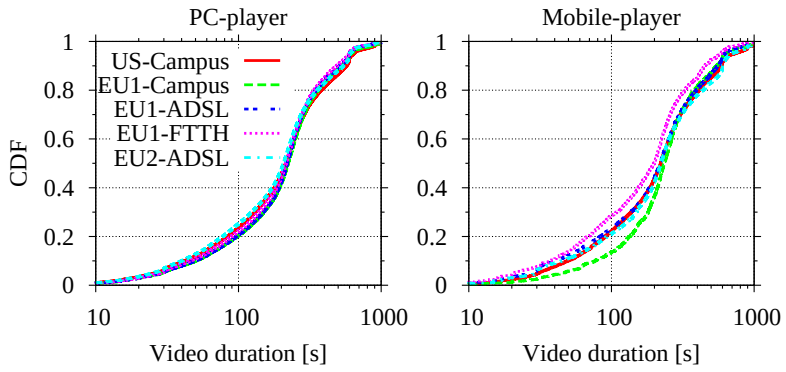


Figure 9.1. CDF of video duration for YT-2011.

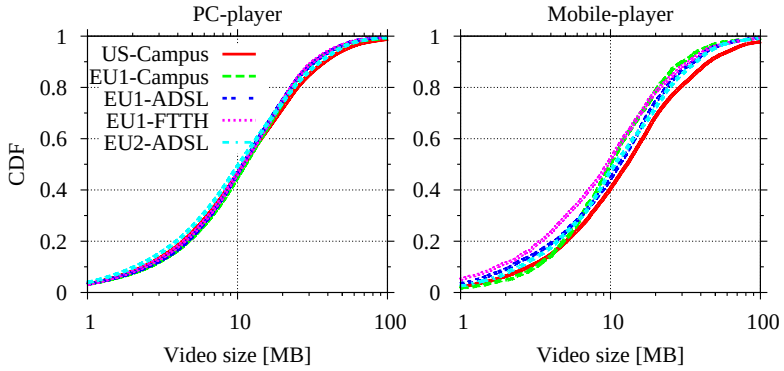


Figure 9.2. CDF of video size for YT-2011.

9.2 Video duration and size

Figure 9.1 reports the duration, in time, of videos for both PC-player (left) and Mobile-player (right). Note that this corresponds to the duration of the complete video and not to the portion of video watched by the user[1]. Considering the PC-player scenario, and comparing the measurements from the different data sets, we notice that there is great similarity across vantage points so that it is impossible to distinguish among them. For example, in all vantage points, 40% of the videos last less than 3 min, and less than 5% of the videos last more than 10 min.

Consider now the Mobile-player case. We observe a slightly moderate difference among the video duration accessed from different probes (notice the log-scale on the x-axis). Still, 40-50% of all videos accessed from mobile terminals are shorter than

[1]This information is extracted from the video metadata.

ID	Video Codec	Audio Codec	Container	Res.	Name
13	H.263	AMR	3GP	144p	others
17	MPEG-4 ASP	AAC	3GP		others
5	FLV1	MP3	FLV	240p	240p-Fl
36	H.264	AAC	3GP		others
34	H.264	AAC	FLV	360p	360p-Fl*
18	H.264	AAC	MP4		360p-Mp+
43	VP8	Vorbis	WebM		others
35	H.264	AAC	FLV	480p	480p-Fl
44	VP8	Vorbis	WebM		others
22	H.264	AAC	MP4	720p	720p-Mp
45	VP8	Vorbis	WebM		others
37	H.264	AAC	MP4	1080p	others
38	H.264	AAC	MP4	3072p	others

(*) PC-player default format , (+) Mobile-player default format

Table 9.1. YouTube supported video formats.

3 minutes, and 5% of videos last more than 10 minutes. Indeed, the Mobile-player and the PC-player CDFs are very similar among them too.

This result shows that people with very different cultural bias (e.g., Europeans vs Americans, students vs residential users), using very different terminals (smartphones vs PCs) and with different Internet access bandwidth (ADSL vs FTTH vs WiFi vs Ethernet) produce and consume the same type of content: short videos which can be quickly watched from YouTube. At the aggregation level that we study, this reflect the distribution of video duration of the YouTube service.

Figure 9.2 reports the total video size in bytes of the videos that have been seen in our data sets. We find very similar results across traces and devices. This is counterintuitive, since we would expect the distribution to be more variable, e.g., due to the availability of videos with different resolutions, and different encoding formats. In addition, intuition would suggest that the video size would be larger for PC-player than for Mobile-player, to better accommodate the limited resources of smartphones. But this is also not clear in the graph. To understand this better, in the next section we dig into the impact of video codecs and resolutions.

9.3 Video formats

A “video” is a complex object that multiplexes encoded video and audio streams. Encoding is done according to different algorithms, and the result is then organized into a container of different type. The combination of the encoding algorithm, video

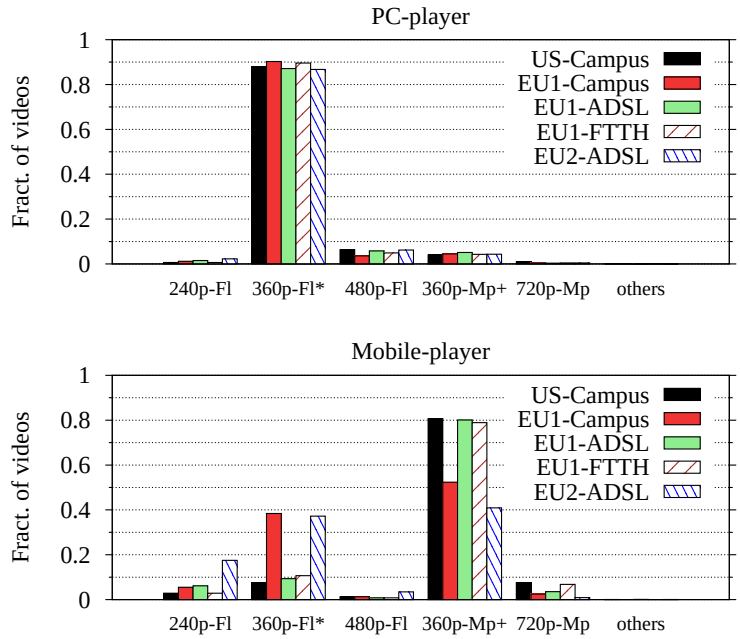


Figure 9.3. Fraction of videos for popular video format for YT-2011.

resolution, and the type of container defines the *video format*. A plethora of video formats are available, some of which are proprietary while others are standard.

YouTube supports the formats listed in Table 9.1. Each format is identified by a unique ID corresponding to the `itag` parameter in the video request. Each ID corresponds to a unique combination of video codec, audio codec, container and resolution. The last column shows the naming convention we used in this paper to identify each format. A marker highlights the YouTube default video format.

The variety of formats reflects the evolution of the system and technology over the last years. In the early days, only Flash Video (FLV) content was supported only at 240p resolution. In 2007 the MP4 container was introduced along with resolution 360p. This switch was driven by the introduction of new devices that did not support FLV videos (e.g., Apple iOS devices). There are also 3GP formats, which are specific for mobile devices, and the more recent WebM formats [75], which are part of the HTML5 specifications. As of today, H.264 video codec is the most widely adopted standard. Note that when the user uploads a new video, the system automatically generates the different video formats and makes them available to download [76].

At playback time, the user can eventually choose among multiple resolutions via the player graphical user interface. For PCs, the Adobe Flash player presents a menu button listing the available resolutions, e.g., 240p, 360p and 480p. Some

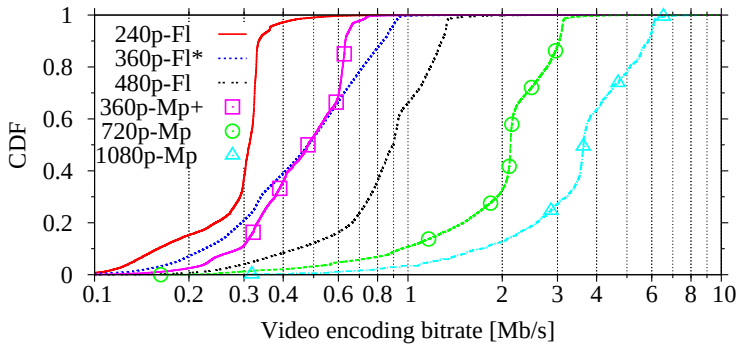


Figure 9.4. CDF of video encoding bitrate for YT-2011.

Mobile-players instead present a toggle button with the choices of "high" and "low" quality, without explicit indication of the resolutions.

The supported formats do not have the same popularity. Figure 9.3 reports the breakdown of video formats considering PC-player and Mobile-player data sets on top and bottom plots, respectively. There is a clear difference respect to the device used to access the video: Flash based formats are largely preferred by PC-player, while MP4 is the preferred container for Mobile-player. This is not surprising considering that Apple iOS products (e.g., iPhone, iPad, iPod touch) cannot handle FLV content. The higher fraction for Flash based formats in the Mobile-player data sets in EU1-Campus and EU2-ADSL may be related to different popularity of devices among certain users (e.g., students) to prefer smartphones running the Android operating system or Windows Mobile which support Flash content.

The default video resolution offered to PC-player is 360p, while Mobile-players tries to retrieve the best available quality according to the network/device capabilities. This causes 720p format (also known as High Definition - HD) to be more popular for Mobile-player than PC-player and this difference is the result of a system design choice.

The previous findings hold true independent from the vantage point, showing the ubiquitousness of the YouTube service. We expect this to change in 3G/4G networks, where the 3GP formats are known to be used and low resolution videos are offered by default.

9.3.1 Video encoding bitrate

Given a codec and a video resolution, the video quality has a strict relation with the *video encoding bitrate*. It is therefore interesting to observe what is the typical encoding bitrate of YouTube videos. Figure 9.4 reports the CDF of the video encoding

bitrate for the most important video formats. Each curve aggregates statistics from all videos of the data sets (each single data set presents the same distribution, being this a system choice). MP4-based formats are highlighted by line-points patterns. In general, the actual encoding bitrate is the minimum between the maximum allowed bitrate, and the bitrate that allows to achieve the desired quality. The latter depends on the video content, e.g., more static video sequences allow to reach lower encoding bitrate. This is reflected in the curves. For example, consider 240p-Fl (FLV) videos. The sharp knee around 300 kbps is the effect of the maximum bitrate limit, which is reached by 70% of videos. About 30% of videos are instead quality limited. Similarly, 360p-Mp (MP4) videos are configured to not exceed 600 kb/s, with most of the videos being quality limited. In some cases, the maximum bound of the video encoding rate can be violated as shown for example for the 10% of 240p-Fl videos. This can be due to a change in the encoding parameters that happened at some time.

It is known that the higher is the resolution, the higher is the bitrate. For example, the 360p videos (currently the default choice) do not exceed 1 Mb/s video rate, while 480p video bitrate goes up to 1.5 Mb/s. 720p and 1080p require up to 3 Mb/s and 6 Mb/s respectively. This allows to speculate on the impact of YouTube switching to higher resolution by default. For example, defaulting to 480p would correspond to almost double the amount of traffic due to YouTube, with possibly large impact on both the YouTube CDN and on ISP networks. Going to 720p as the default choice would correspond to multiply by a factor of 4 the offered traffic. Given that YouTube already accounts for more than 20% of Internet traffic and assuming the user demand remains the same, this would correspond to a critical traffic surge that might impair the YouTube service itself.

Chapter 10

Video server selection strategies

10.1 Introduction

In this chapter, we focus on the YouTube infrastructure aiming at exploring the various factors that can influence the video delivery process, such as user proximity to data centers, server load, and popularity of video content. Recently, a few works analyzed the YouTube video delivery infrastructure [1, 66]. However, both these works focus on the "old" YouTube infrastructure. In [1], the authors discovered eight data centers around the U.S. that provided most videos to clients around the world. Further, they found that the YouTube server selection algorithm does not consider geographical location of clients and that requests are directed to data centers proportionally to the data center size. In [66] the authors perform PlanetLab experiments to download YouTube videos and measure user performance. The authors found that most videos are being sent from a few locations in the U.S. and that YouTube pushes popular videos to more data centers.

Conversely, our work focuses on the "new" YouTube infrastructure. Based on the analysis of YT-2010 data set (see Chapter 8.2.1), a much robust and heterogeneous data set with respect to the cited works, and the usage a state-of-the-art geolocalization algorithm, we found several differences with respect to previous works. First of all, 33 different data centers have been found serving YouTube videos all around the world. Analyzing RTT distances, we found that the video delivery is *location aware*, i.e., most videos are being delivered from a *preferred data center*, typically the closest one. However, a combination of different causes related to DNS and caching policies can *redirect* the download to non-preferred data centers.

The remaining of this chapter is organized as follows. We start studying the video server geolocation, introducing the problem, the state-of-the-art tools and the methodology we developed to group the servers in data centers. Leveraging on this information, we then analyze the server selection strategies dissecting the

Data set	AS 15169 Google Inc.		AS 43515 YouTube-EU		Same AS		Others	
	servers	bytes	servers	bytes	servers	bytes	servers	bytes
US-Campus	82.8	98.96	15.6	1.03	0	0	1.4	0.01
EU1-Campus	72.2	97.8	20.3	1.6	0	0	7.5	0.6
EU1-ADSL	67.7	98.8	28	0.94	0	0	4.3	0.26
EU1-FTTH	70.8	99	24.2	0.83	0	0	5	0.27
EU2-ADSL	62.9	49.2	28.6	10.4	1.1	38.6	7.4	1.8

Table 10.1. Percentage of servers and bytes received per AS for YT-2010.

possible causes of redirection: DNS resolvers, cache miss, load balancing, and video popularity.

10.2 Video server geolocation

In this section we study the video server geolocation. The goal is to later use this information to analyze the video server selection policies. We start the analysis showing how the traffic is split with respect to Autonomous Systems (AS). Then, we introduce the geolocalization problem and the developed methodology to aggregate IP addresses in data center. Finally, we present the results obtained.

10.2.1 AS location

We start our analysis studying the Autonomous System (AS) in which YouTube video servers are located. We employ the `whois` tool to map the server IP address to the corresponding AS. Table 10.1 presents our findings for each data set. The second group of columns shows the percentage of servers and bytes sent from the Google AS (AS 15169). Not surprisingly, most servers are hosted in the Google AS. For instance, for the US-Campus data set, 82.8% of the servers are located in the Google Inc. AS, serving 98.66% of all bytes. The third group of columns shows that a small percentage of servers (and an even smaller percentage of bytes) are still located in the YouTube-EU AS (AS 43515). We therefore have an evidence that since 2009 Google has migrated most content from the YouTube original infrastructure (that was based on third party CDNs) to its own CDN. The traffic served from the YouTube networks is probably because of legacy configurations. This contrasts with earlier studies such as [66, 1], according to which the majority of servers were located in the YouTube AS (AS 36561, now not used anymore).

The fourth group of columns in Table 10.1 shows the percentage of servers and

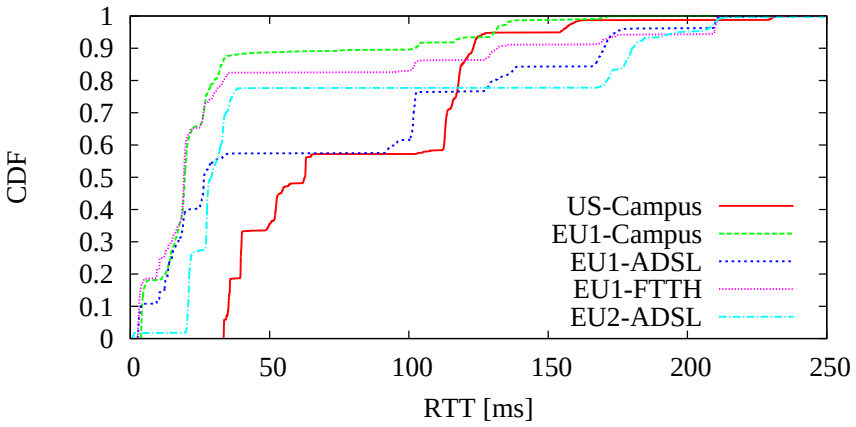


Figure 10.1. RTT to YouTube content servers from each vantage point considered in YT-2010.

bytes received from within the same AS where the data set have been collected. Note that the values are 0 for all data sets except EU2-ADSL. The EU2-ADSL data set indeed shows that a YouTube data center is present inside the ISP network. This data center serves 38.6% of the bytes in the EU2-ADSL data set. This results in the EU2-ADSL data set having fairly different performance than other data sets, as our analysis will reveal later.

Finally, the last groups of columns aggregates the percentage of servers and bytes sent from other ASes, among which CW (AS1273) and GBLX (AS3549) are the most likely one. This confirms therefore that YouTube servers can be both present inside an ISP, or in the Google network.

In the rest of this chapter, we only focus on accesses to video servers located in the Google AS. For the EU2-ADSL data set, we include accesses to the data center located inside the corresponding ISP.

10.2.2 Limitations of IP-to-location databases

One common way to find the geographical location of an IP address is to rely on public databases [66]. While such databases are fairly accurate for IPs belonging to commercial ISPs, they are known to be inaccurate for geolocation of internal IPs of large corporate networks. For example, according to the Maxmind database [50], all YouTube content servers found in the data sets should be located in Mountain View, California, USA. To verify this, we perform RTT measurements from each of our vantage points to all content servers found in our data sets. Figure 10.1 reports the Cumulative Distribution Function (CDF) of the minimum RTT obtained to each server. We clearly observe that there is a lot of variation in the measurements, and

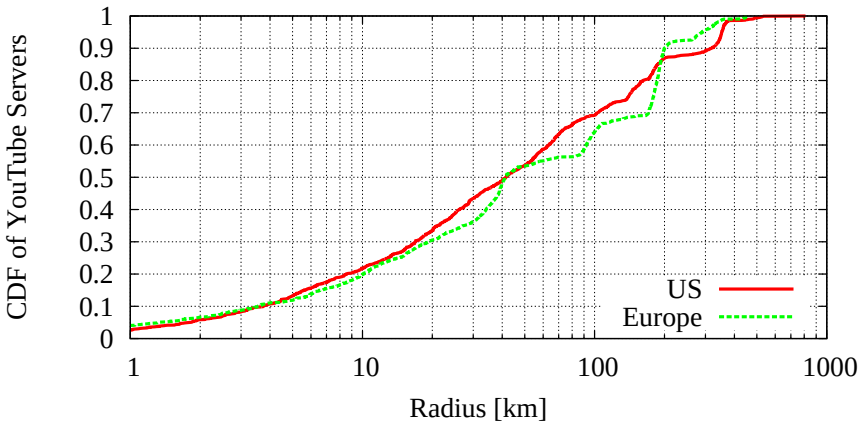


Figure 10.2. Radius of the CBG confidence region for the YouTube servers found in YT-2010.

in particular, many of the RTT measurements for the European connections are too small to be compatible with intercontinental propagation time constraints [61]. This indicates that all servers cannot be located in the same place.

We note that Maxmind was useful in [66], probably because most YouTube servers in the old infrastructure were reported as located in San Mateo and Mountain View, California, USA. Further, a recent work [1] adopts a different approach, where the location of the server is obtained directly from the server name. However, this approach is not applicable to the new YouTube infrastructure, where DNS reverse lookup is not allowed. Therefore we decided to adopt a measurement-based approach to systematically localize YouTube servers.

10.2.3 Measurement based geolocation mechanism

CBG [31] is a well-known geolocation algorithm that is based on simple triangulation. A set of landmarks is used to measure the RTT to a target. A simple linear function is then used to estimate the physical distance between each landmark and the target. This distance will become the radius of a circle around the landmark where the target must be located. The intersection among all circles is the area in which the target can be located.

We obtained the CBG tool from Gueye et al. [31] for our evaluations. We used 215 PlanetLab nodes as landmarks: 97 in North America, 82 in Europe, 24 in Asia, 8 in South America, 3 in Oceania and 1 in Africa. Then, we run RTT measurements from each landmark to each of the YouTube servers that have been found in our data set, and identified the area in which they are placed.

In Figure 10.2 we evaluate the confidence region of CBG, i.e. the area inside

Data set	N. America	Europe	Others
US-Campus	1464	112	84
EU1-Campus	82	713	1
EU1-ADSL	518	769	51
EU1-FTTH	90	631	44
EU2-ADSL	233	815	0

Table 10.2. Google servers per continent on each data set in YT-2010.

which the target IP should be located. The picture shows the CDF of the radius of the confidence region for all servers found. Separate curves are shown for IPs in U.S. and Europe. Note that the median for both U.S. and European servers is 41km, while the 90th percentile is 320km and 200km, respectively. This is in the ballpark of the PlanetLab experiments presented in [31], where the 90th percentile for U.S. and Europe was about 400km and 130km. We can therefore consider the results provided by CBG to be more than adequate for our analysis.

10.2.4 Geolocation results

Table 10.2 details the result of using CBG to identify the location of all the destination IPs found in the data sets. The table shows the number of servers that are located in North America, Europe and other continents. Interestingly in each of the data sets, at least 10% of the accessed servers are in a different continent.

Finally, since several servers actually fall in a very similar area, we consider all the YouTube servers found in all the data sets and aggregate them into the same "data center". In particular, servers are grouped into the same data center if they are located in the same city according to CBG. We note that all servers with IP addresses in the same /24 subnet are always aggregated to the same data center using this approach. We found a total of 33 data centers in our data sets, 14 in Europe, 13 in USA and 6 in other places around the world. These results may not cover the complete set of YouTube servers since we are only considering those servers that appeared in our data set.

10.3 Evaluating YouTube's server selection algorithm

In the previous section, we have shown how IP addresses of YouTube servers may be mapped to the appropriate YouTube data centers. Armed with such information, we now try to understand how user video requests are mapped to YouTube data

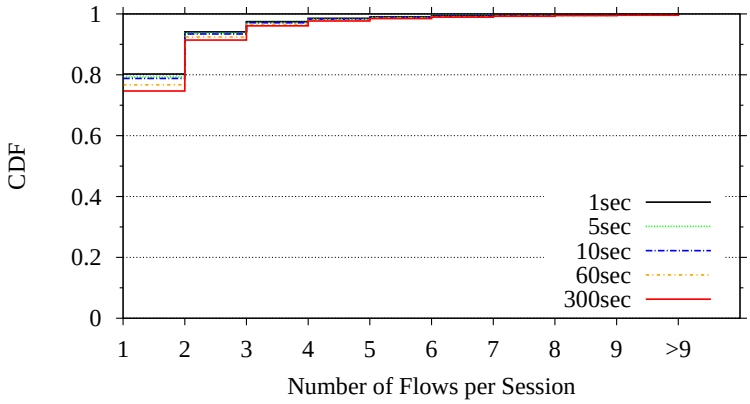


Figure 10.3. Number of flows per session with different values of T for the US-Campus in YT-2010.

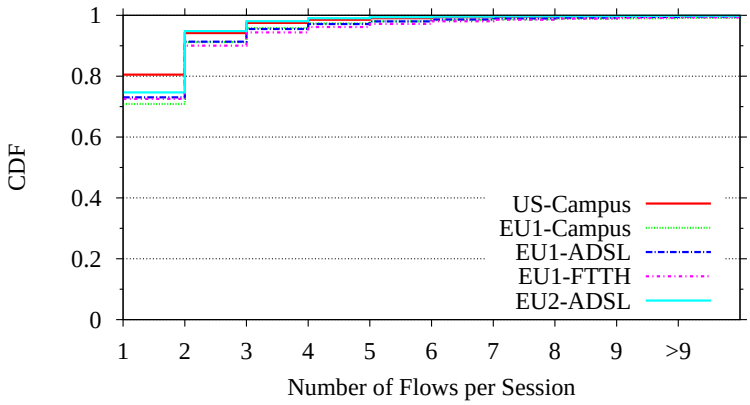


Figure 10.4. Number of flows per session for all data sets using T=1 second in YT-2010.

centers. We are interested in exploring the various factors that can influence the decision, such as user proximity, server load, and popularity of content. We begin by discussing the various types of flows in a YouTube session, and then discuss how content servers are selected.

10.3.1 Video sessions tuning

As introduced in Chapter 7.2, when a user attempts to download a video, the overall interaction may include a group of distinct flows, not all of which involve transfer of video. To group the flows in sessions, we need to select the value of the threshold T which defines if two flows are overlapped or not (see Chapter 8.3). Since we are interested in capturing interactions triggered by the system (e.g., redirections), we want to use a small value of T, but that is large enough to avoid artificially separated flows. Hence, we perform sensitivity to the value of T in our traces. We show results for the US-Campus data set in Figure 10.3 and note that other traces show similar

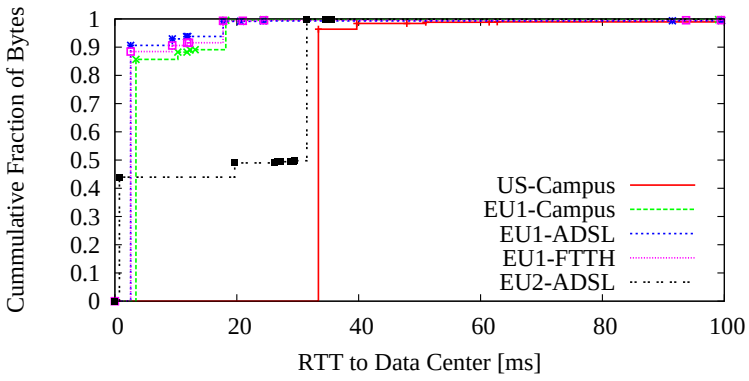


Figure 10.5. Fraction of the total YouTube video traffic served by a data center with an RTT less than a given value in YT-2010.

trends. Results indicate that values of T equal to 10 seconds or less generate similar number of sessions. So we pick the smallest value of T in our evaluations, $T = 1$ second.

Figure 10.4 reports the CDF of the number of flows per session for each data set, assuming $T = 1$ second. It shows that $72.5 - 80.5\%$ of the sessions consist of a single (long) flow. Therefore, normally there is no need to iterate over different servers to download the video data. However, $19.5 - 27.5\%$ of the sessions consist of at least 2 flows, showing that the use of application-layer redirection is not insignificant. This is also confirmed by the sharp knees in the [800-1000] B range in Figure 10.3 corresponding to the `'302 Found'` messages which are triggered when a redirection occurs.

10.3.2 Understanding server selection strategy

In Table 10.2 we have shown that the users in each data set contact video servers all over the world. It is now interesting to investigate how the volume of traffic downloaded is spread across the different data centers. Figure 10.5 reports the fraction of traffic served by each data center versus the RTT between the vantage points and the data centers itself. In particular, we consider the minimum RTT seen by pinging all servers in each data center from the probe PC installed in the PoP. We observe that except for EU2-ADSL, in each data set one data center provides more than 85% of the traffic. We refer to this primary data center as the *preferred* data center for that particular trace and other data centers will be labeled as *non-preferred*. At EU2-ADSL, two data centers provide more than 95% of the data, one of them located inside the ISP and the other outside in the Google AS. We label the data center with the smallest RTT in EU2-ADSL as the preferred one. We give

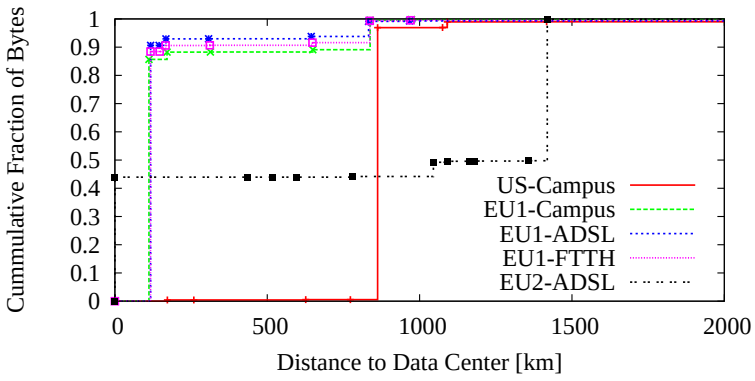


Figure 10.6. Fraction of the total YouTube video traffic served by a data center with a distance less than a given value in YT-2010.

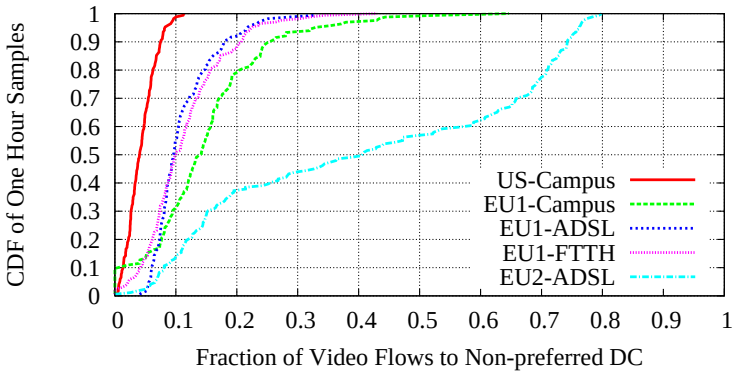


Figure 10.7. Variation of the fraction of video flows directed to a non-preferred data center over time. One hour long time periods are considered in YT-2010.

a closer look to the EU2-ADSL case in Chapter 10.4.1.

Further, we notice that the data center that provides most of the traffic is also the data center with the smallest RTT for each data set. This suggests that RTT does play a role in the selection of YouTube servers. However, we have reason to believe that RTT is not the only criteria and that the preferred data center may change over time. For example, in a more recent data set collected in February 2011, we found that the majority of US-Campus video requests are directed to a data center with an RTT of more than 100 ms and not to the closest data center, which is around 30 ms away.

Figure 10.6 considers the distance (in kilometers) between users and the data centers they are mapped to. In most cases, the data centers with the smallest delay

to the customers are also the physically closest ones. This is not the case for the US-Campus data set, where the five closest data centers provide less than 2% of all the traffic. Coupled with previous observations about RTT, this is an indication that geographical proximity is not the primary criterion used in mapping user requests to data centers.

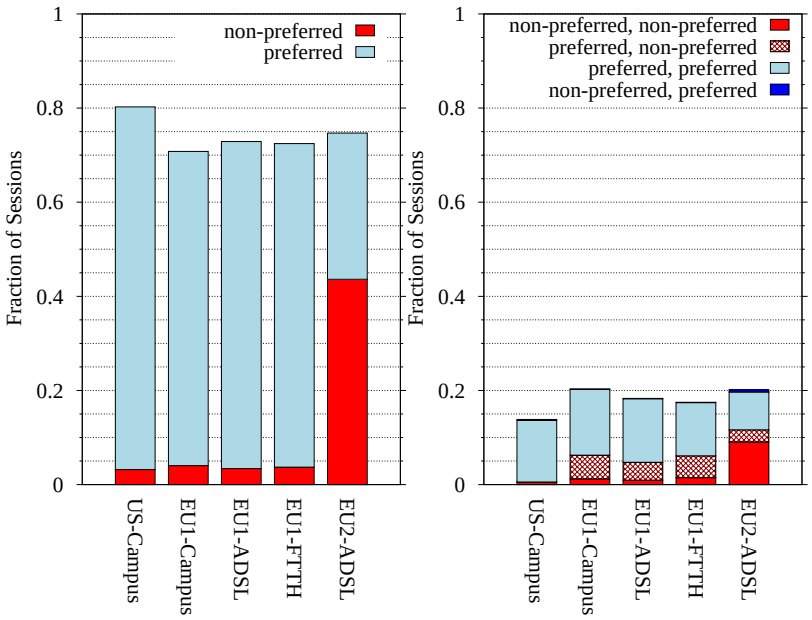
The final observation we make is that although most traffic comes from the preferred data center that is typically very close to the customers, there are some exceptions in all data sets. For the US-Campus and the EU1 data sets, between 5% and 15% of the traffic comes from the *non-preferred* data centers. However, in EU2-ADSL, more than 55% of the traffic comes from *non-preferred* data centers. We now are interested to see the variation over time of the fraction of traffic coming from non-preferred data centers. One hour-long time slots are considered, and the fraction of traffic served by non-preferred data centers in each of these time slots is determined. Figure 10.7 plots a CDF of these fractions. The results indicate that the fraction varies across time for most data sets, the variation being most prominent for the EU2-ADSL data set. In particular for this data set, 50% of the samples have more than 40% of the accesses directed to the non-preferred data center.

10.3.3 Mechanisms resulting in accesses to non-preferred data centers

We have seen that a non-negligible fraction of video flows are downloaded from non-preferred data centers. There are at least two possible causes for this. A first possibility is that the DNS mechanisms direct a request to the non-preferred data center. A second possibility is that the request was redirected to another data center by the preferred data center server.

To disambiguate the two cases, we consider the video session associated with each flow, as discussed in Chapter 10.3.1. In the case that DNS maps a request to a non-preferred data center, the video session must consist of a single video flow to a non-preferred data center, or must begin with a control flow to the non-preferred data center. In the other scenario, the session must begin with a control flow to the preferred data center (indicating the DNS mapping was as expected), but subsequent flows in the session must be to non-preferred data centers.

To better understand the effectiveness of DNS in mapping requests to the preferred data center, consider Figure 10.8(a). Each bar in the figure shows the fraction of sessions that involve only one flow. Further, each bar shows a break down of the requests sent to the preferred and non-preferred data centers. For instance, for US-Campus, 80% of the sessions involve a single flow; 75% are then served by the preferred data center while 5% of sessions are directly going to the non-preferred data center. Interestingly, about 5% of the single-flow sessions are directly served by



(a) 1 flow sessions (b) 2 flows sessions

Figure 10.8. Breakdown of sessions based on whether flows of the session are sent to preferred data center in YT-2010.

the non-preferred data center for EU1 data sets too. For EU2-ADSL however, over 40% of the single flow sessions are served by the non-preferred data center. Overall, these results show that DNS is in general effective in mapping requests to the preferred data center. Still DNS mapping mechanisms do account for a significant fraction of video flow accesses to non-preferred data centers.

We next try to understand the extent to which users downloaded video from a non-preferred data center, even though they were directed by DNS to the preferred data center. Figure 10.8(b) presents the breakdown of sessions involving 2 flows. These sessions group a control flow followed by a video flow. Based on whether each flow involves the preferred or non-preferred data center, we have four possible cases: (i) both preferred; (ii) both non-preferred; (iii) the first preferred and the second non-preferred; and (iv) the first non-preferred and the second preferred. Each bar in Figure 10.8(b) presents the breakdown among these patterns. For all the EU1 data sets, we see a significant fraction of cases where the DNS did map requests to the preferred data center, but application-layer redirection mechanisms resulted in the user receiving video from a server in a non-preferred data center. For the EU2-ADSL data set, we note that a larger fraction of sessions has both flows going to the non-preferred data center, meaning that the DNS is still the primary cause for the

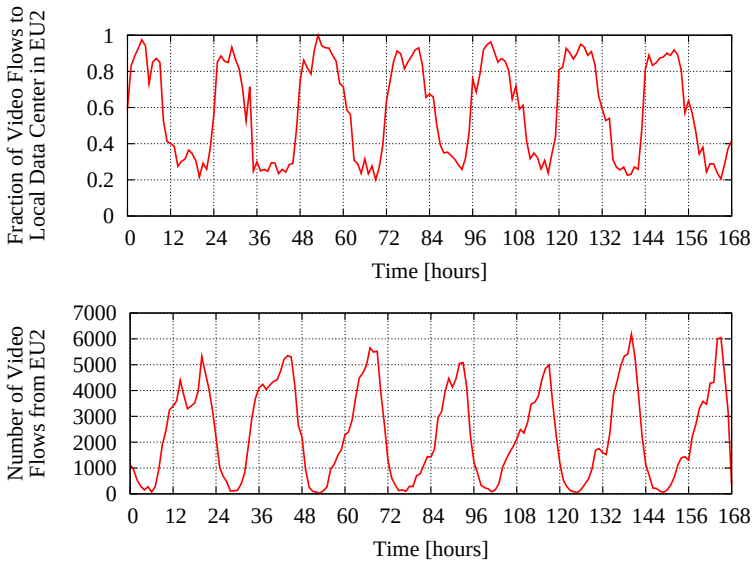


Figure 10.9. Fraction of the total YouTube video traffic served by the preferred data center (top graph) and total number of video flows (bottom graph) as a function of time for the EU2-ADSL data set in YT-2010.

user downloading videos from non-preferred data centers. We have also considered sessions with more than 2 flows. They account for $5.18 - 10\%$ of the total number of sessions, and they show similar trends to 2-flow sessions. For instance, for all EU1 data sets, a significant fraction of such sessions involve their first access to the preferred data center, and subsequent accesses to non-preferred data centers.

10.4 Causes underlying non-preferred data center accesses

10.4.1 DNS-level load balancing

As shown in the previous Section, the EU2-ADSL data set exhibits very different behavior compared to other data sets. Over 55% of the video traffic is received from the non-preferred data center, and a vast majority of accesses to non-preferred data centers is due to the DNS mapping mechanisms.

To understand this better, consider Figure 10.9. The top graph presents the evolution over time of the fraction of video flows served by the preferred data center. One hour time slots are considered. The bottom graph shows the total number of video flows seen in the EU2-ADSL data set as a function of time. Note that time 0 represents 12am on Friday. We can clearly see that there is a day/night pattern in

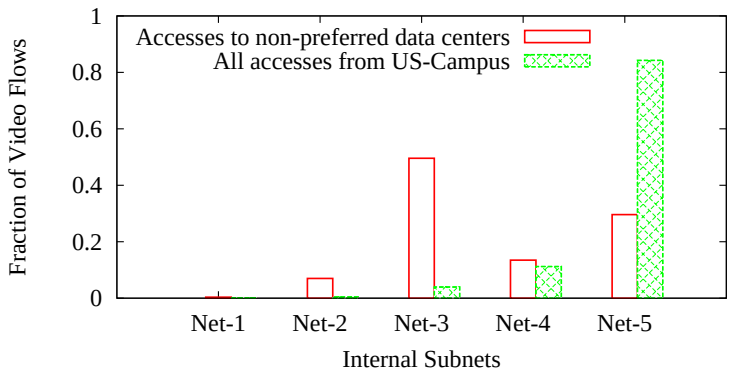


Figure 10.10. Fraction of all video flows, and video flows to non-preferred data centers for each internal subnet of the US-Campus in YT-2010.

this set of requests. During the night, when the total number of accesses from EU2-ADSL is small, the internal data center handles almost 100% of the video requests. However, during the day, when the number of requests per hour goes up to around 6000, the fraction of requests handled by the local data center is always around 30% across the whole week. Results for other data sets are not shown for the sake of brevity. Still, all data sets exhibit a clear day/night pattern in the number of requests. However, there is less variation over time of the fraction of flows served by the preferred data center, as already seen in Figure 10.7. Furthermore, there is much less correlation with the number of requests.

We believe the reason for this is the unique setup in the EU2-ADSL network. In this network, the data center inside the network serves as the preferred data center. While this data center located inside the ISP is the nearest to the users, it is unable to handle the entire load generated by users inside the EU2-ADSL ISP during busy periods. There is strong evidence that adaptive DNS-level load balancing mechanisms are in place, which results in a significant number of accesses to the non-preferred data centers during the high load period of traffic.

10.4.2 Variations across DNS servers in a network

Our results from the previous section indicate that for the US-Campus data set most of the accesses to the non-preferred data center are caused by DNS. Deeper investigation indicates that most of these accesses may be attributed to clients from a specific internal subnet within the US-Campus network. Those clients indeed request significantly higher fraction of videos from non-preferred data centers than clients in other subnets. To see this, consider Figure 10.10. Each set of bars corresponds to an internal subnet at US-Campus. The bars on the left and right respectively

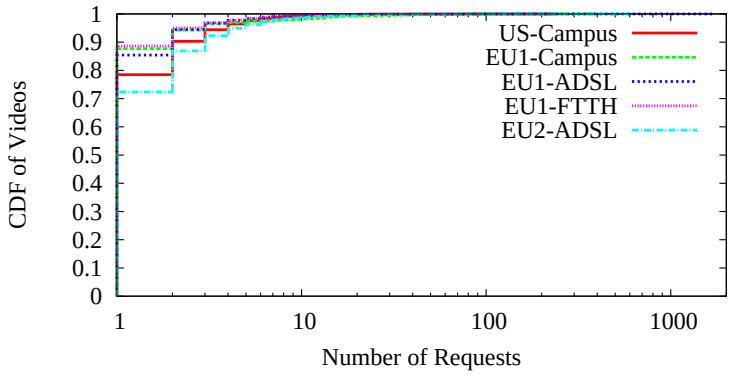


Figure 10.11. Number of requests for a video to the non-preferred data centers in YT-2010.

show the fraction of accesses to non-preferred data centers, and the fraction of all accesses, which may be attributed to the subnet. Net-3 shows a clear bias: though this subnet only accounts for around 4% of the total video flows in the data set, it accounts for almost 50% of all the flows served by non-preferred data centers.

Further investigation shows that hosts in the Net-3 subnet use different DNS servers that map YouTube server names to a different preferred data center. In other words, when the authoritative DNS servers for the YouTube domain are queried by the local DNS servers in Net-3, the mapping provided is to a different preferred data center than the other subnets on US-Campus. We believe this behavior is not a misconfiguration in the YouTube servers or the Net-3 servers, but we rather hypothesize that this is the result of a DNS-level assignment policy employed by YouTube, probably for load balancing purposes, which can vary between DNS servers and thus subnets that belong to the same campus or ISP network.

10.4.3 Investigating redirection at the application layer

We now consider cases where users download video from non-preferred data centers, even though DNS mapped them to the preferred data center.

To get more insights into this, consider Figure 10.11 which reports the CDF of the number of times a video is downloaded from a non-preferred data center. Only videos that are downloaded at least once from a non-preferred data center are considered. The results show two trends. First, a large fraction of videos are downloaded exactly once from the non-preferred data center. For example, for the EU1-Campus data set, around 85% of the videos are downloaded only once from the non-preferred data center. Second, there is a long tail in the distributions. In fact, some videos are downloaded more than 1000 times from non-preferred data centers.

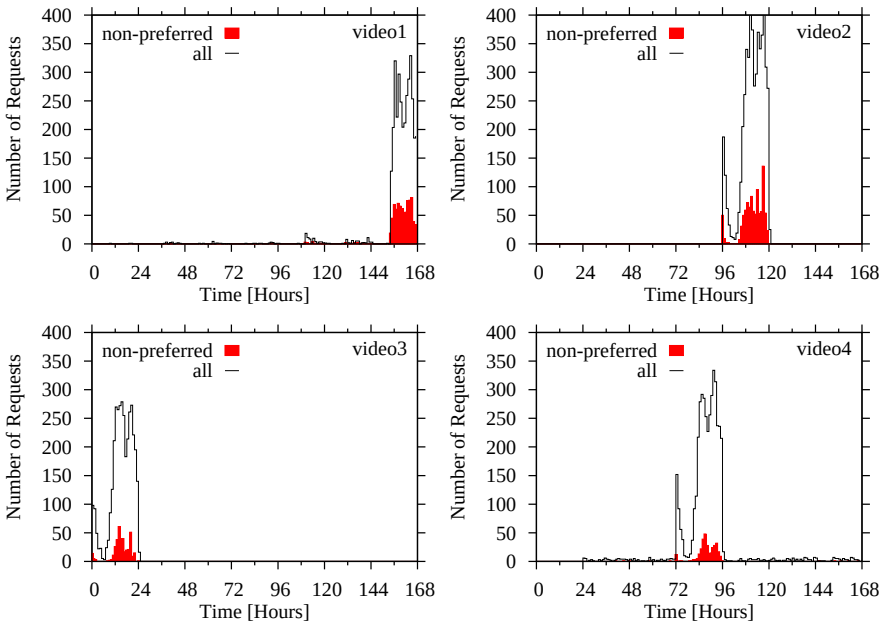


Figure 10.12. Load related to the top 4 videos with the highest number of accesses to the non-preferred data centers for the EU1-ADSL in YT-2010.

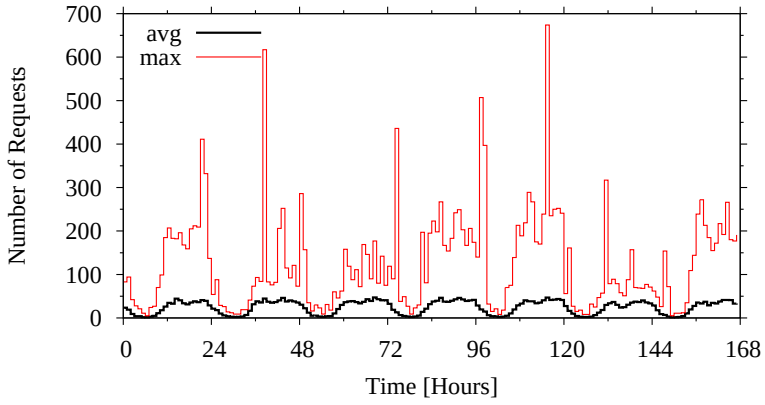


Figure 10.13. Average and maximum number of requests per server in the preferred data center of EU1-ADSL in YT-2010.

We consider the impact of popular and unpopular videos on server selection in the next few paragraphs.

Alleviating hot-spots due to popular videos: Let us focus first on the tail in Figure 10.11. Figure 10.12 considers the four videos with the highest number of accesses to the non-preferred data centers for the EU1-ADSL data set. Each

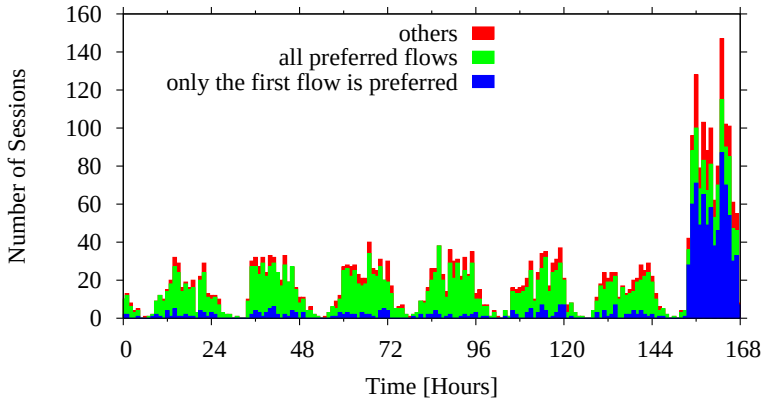


Figure 10.14. Number of video sessions per hour seen by the server handling *video1* in the preferred data center of the EU1-ADSL in YT-2010. A breakdown of sessions based on whether flows are directed to preferred data centers is also shown.

graph corresponds to one of the videos, and shows (i) the total number of accesses to that video; and (ii) the number of times the video is downloaded from the non-preferred data center, as a function of time. We see that there are spikes indicating that some videos are more popular during certain limited periods of time. Most accesses to non-preferred data centers occur during these periods. In particular, all these videos were played by default when accessing the `www.youtube.com` web page for exactly 24 hours, i.e., they are the "video of the day".

Those are therefore very popular videos, which possibly generate a workload that can exceed the preferred data center capacity. Therefore, application-layer redirection is used to handle the peaks. As further evidence, Figure 10.13 shows the average and the maximum number of requests served by each server (identified by its IP address) in the preferred data center as a function of time. The figure shows that at several times, the maximum number of requests a single server has to handle is by far larger than the average load. For instance at time 115, the average load is about 50 video flows, but there is one server that answers more than 650 requests. Interestingly, we note that the servers suffering the peak loads are those serving the majority of the top videos of Figure 10.12.

Further investigation reveals that DNS correctly forwards the request to a server in the preferred data center, but since its load is too high, the server redirects part of the requests to another server in a non-preferred data center. Consider Figure 10.14, which shows the load in terms of sessions, handled by the server receiving the requests for *video1* for the EU1-ADSL data set.

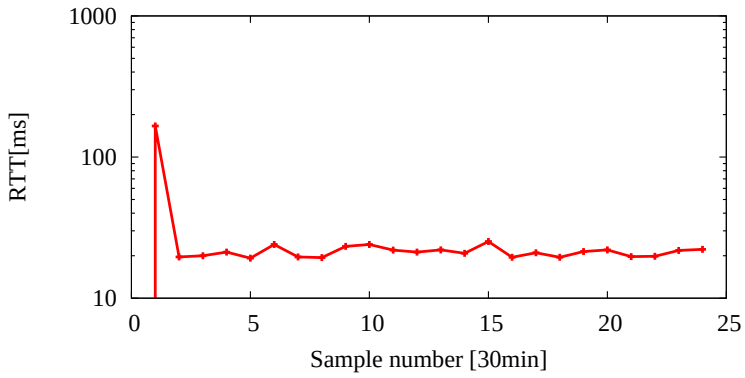


Figure 10.15. Variation over time of the RTT between a PlanetLab node and the content servers for requests of the same test video.

Different colors are used to show the breakdown of the total number of sessions according to the preferred/non-preferred patterns. For example, we can see that in the first 6 days, the majority of the sessions involves only flows served by the preferred data center. On the last day however, a larger number of requests is received, which leads to an increase in application-layer redirections to a non-preferred data center. Overall, these results show that local and possibly persistent overload situations are handled by the YouTube CDN via application-layer redirection mechanisms.

Availability of unpopular videos:

Consider again Figure 10.11. Let us now focus on the observation that several videos are downloaded exactly once from the non-preferred data center. Further analysis indicated that for most data sets, over 99% of these videos were accessed exactly once in the entire data set, with this access being to non-preferred data centers. However, when the videos were accessed more than once, only the first access was redirected to a non-preferred data center.

This observation leads us to hypothesize that downloads from non-preferred data centers can occur because of the limited popularity of the videos. In particular, videos that are rarely accessed may be unavailable at the preferred data center, causing the user requests to be redirected to non-preferred data centers until the video is found.

Since our data sets only contain a limited view of the accesses seen by a data center, it is difficult to validate this claim using only our data sets. We therefore conducted controlled active experiments using PlanetLab nodes. In particular, we uploaded a test video to YouTube. The video was then downloaded

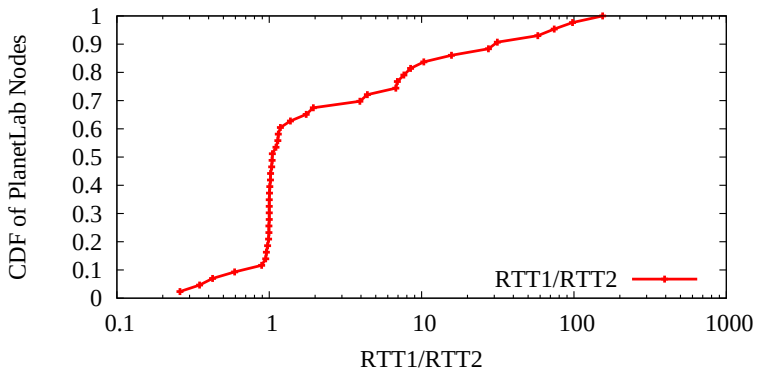


Figure 10.16. Reduction in RTT from PlanetLab nodes to the content servers when a test video is downloaded twice. The first access may incur a higher RTT due to unavailability of content in the preferred data center.

from 45 PlanetLab nodes around the world. Nodes were carefully selected so that most of them had different preferred data centers. From each node, we also measured the RTT to the server being used to download the content. We repeated this experiment every 30 minutes for 12 hours.

Figure 10.15 shows an example of the variation of RTT samples considering a PlanetLab node located in California. Observe that the very first sample has an RTT of around 200 ms. In contrast, later samples exhibit RTT of about 20 ms. Further investigations showed that the first time, the video was served by a data center in the Netherlands while subsequent requests were served by a data center in California.

Figure 10.16 shows the CDF of the ratio of the RTT to the server that handled the first video request (RTT1) to the RTT to the server that handled the second video request (RTT2) for all the PlanetLab nodes. A ratio greater than 1 means that the video was obtained from a closer data center in the second attempt than in the first attempt. A ratio with a value close to 1 shows that the first request went to the same server or a server in the same data center as the second request. For over 40% of the PlanetLab nodes, the ratio was larger than 1, and in 20% of the cases the ratio was greater than 10. Interestingly, we have also found the RTT of subsequent samples is comparable to the RTT of the second sample. Overall, these results indicate that the first access to an unpopular video may indeed be directed to a non-preferred data center, but subsequent accesses are typically handled from the preferred data center.

10.5 Conclusion

The YouTube CDN has been completely redesigned compared to the one previously analyzed in the literature. In the new design, most YouTube requests are directed to a preferred data center and the RTT between users and data centers plays a role in the video server selection process. More surprisingly, however, our analysis also indicates a significant number of instances (at least 10% in all our datasets) where videos are served from non-preferred data centers. In most datasets at least 10% of requests were not served from the preferred data center. In one of our datasets, up to 55% of video requests were not served from the preferred data center, while in most datasets at least 10% of requests were not served from the preferred data center. In many of our datasets, there were some time periods when a much larger fraction of requests to non-preferred data centers was observed. Across all datasets, a large fraction of videos are downloaded no more than once from the non-preferred data center, however there is a long tail with videos being downloaded more than 1000 times from non-preferred data centers. We identified a variety of causes underlying accesses to non-preferred data centers including: (i) load balancing; (ii) variations across DNS servers within a network; (iii) alleviation of load hot spots due to popular video content; and (iv) availability of unpopular video content in a given data center. Overall these results point to the complexity of factors that govern server selection in the YouTube CDN.

The adoption of *video sessions* is crucial to capture the complete evolution of a download and we identified a variety of causes underlying accesses to non-preferred data centers. In some cases DNS mechanisms resulted in requests being directed to non-preferred data centers for reasons including (i) load balancing; (ii) variations across DNS servers within a network; to handle variations in system load due to the day/night patterns in YouTube usage; and (ii) variations across DNS servers within a network. Interestingly, we found other cases where videos were served from a non-preferred data center because of redirections at the application layer. videos were served from a non-preferred data center, even though DNS directed the user to the preferred data center. The common causes underlying such cases included (i) alleviation of load hot spots due to popular video content; and (ii) accesses to unpopular video content that may not be available in a given data center. Overall these results point to the complexity of factors that govern server selection in the YouTube CDN.

Chapter 11

User behaviour and implications

In this chapter we focus our attention on the way people watch videos from the YouTube system. In the literature, the majority of the works characterizing the users' behaviour are focused on the activities related to the YouTube portal. For example, using crawling methodologies, [12] study the social network related to the videos, while [8] characterizes the different type of uploaders. However, very little is known on the interaction of the users with the YouTube player, if the users perform resolution switches or watch the whole video.

Considering mobile devices, recently there have been several works characterizing high level usage patterns [49, 27, 70]. [49] shows that the number of mobile devices doubled between 2009 and 2010 and that more than 80% of mobile devices traffic is HTTP, with multimedia traffic alone accounting for more than 30% of HTTP. [27] compares the content and flow characteristics of mobile devices and PCs traffic. Using a DPI tool, the authors are able to show that YouTube alone accounts for more than 35% of the Internet traffic. Anyway, these works do not specifically address the YouTube system.

In this work we aim at a much deeper characterization of the users' behavior accessing YouTube from both PCs and mobile devices. We are interested in observing if they interact with the player's GUI, e.g., switching resolution or going in full screen mode, and which portion of the video people actually watches. Both have interesting implication on the workload the system has to handle and the efficiency it achieves in serving the requests. We take advantage of the YT-2011 data set and we conducted an in depth analysis of the users' behaviour and the implications related to the users' actions. Moreover, we consistently compare PC-player and Mobile-player download and we highlight problems caused by the YouTube infrastructure when delivering videos to mobile devices.

Results show that users stick to the default playback parameters and they are not interested in changing the resolution during the playback. Moreover, typically they consume just a portion of the video. Given the aggressive buffering scheme adopted

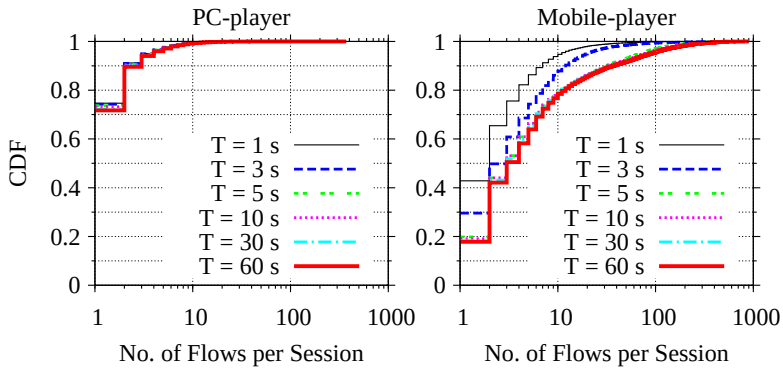


Figure 11.1. Sensitivity of the number of TCP connections per session for different values of T. EU1-ADSL data set in YT-2011.

by YouTube, this behaviour leads to waste a non negligible amount of bandwidth.

In the remaining of this chapter, we start from the methodology we used to study the traffic and the tuning the T threshold to define the *video sessions*. We then focus on the resolution switch, investigating on at which time it is performed and which are the most common resolutions used in the switch. Finally, we study the fraction of video downloaded and the implications with respect to the buffering policies used by YouTube.

11.1 Video session tuning

To study the evolution of the downloads, we need to properly tune the threshold T as obtain the video sessions (see Chapter 8.3). Respect to Chapter 10, where we were interested in studying the automatic interactions generated by the system, in this case we work at user-level so $T > 1$ sec is appropriate. Figure 11.1 reports the number of connections per session for different values of T. The EU1-ADSL data set is considered, but other data sets show identical results. The choice of T is not critical for PC-player, while $T > 5$ s is required to properly aggregate Mobile-player connections. In the following, we set $T = 60$ s, a conservative choice to better capture users actions that could happen after the download has been completed but while the playback is still running.

Figure 11.1 also shows the impact of the Mobile-player mechanisms in the number of connections per session. While for PC-player, only 2% of the sessions have more than 6 connections, for Mobile-player more than 4% of the sessions involve more than 100 connections.

	PC-player			Mobile-player		
Data set	0	1	>1	0	1	>1
US-Campus	95.10	4.60	0.30	99.75	0.19	0.05
EU1-Campus	96.62	3.12	0.27	99.28	0.61	0.10
EU1-ADSL	95.27	4.45	0.28	99.63	0.28	0.09
EU1-FTTH	95.73	3.99	0.28	99.39	0.42	0.19
EU2-ADSL	95.14	4.40	0.46	98.07	1.36	0.57

Table 11.1. Percentage of resolution switch in YT-2011.

11.2 Resolution switch

In Chapter 9.3 we have seen that YouTube supports different resolution, each associated to a standard numerical code (360p, 480p, etc.) which is specified by the client interacting with the video server. These codes are listed in a menu of the player GUI and the user can change resolution during the playback. Intuitively, the higher the value, the better the resolution. When a resolution switch occurs, the current video flow is aborted and a new one with the new resolution specified is initiated. It follows that, given a video session, a resolution switch is easily detected by observing requests with the same `videoID`, but different video format.

Note that, as described in Chapter 9.3, the same resolution can be served with both Flash and Mpeg container. This characteristic is hidden to the user which receives the format according to the characteristics of the device he is using (e.g., Flash for PC-player and Mpeg for Mobile-player). However, as to disambiguate these cases, we label each video format appending the first two letter of the video container to the standard resolution code. For example, 360p-Fl corresponds to a 360p resolution with the Flash container.

Table 11.1 reports the percentage of sessions involving zero, one or more than one resolution switch for both PC-player and Mobile-player. Surprisingly, results show that a resolution switch happens for less than 5% of PC-player sessions. This is an example of inertia, where users stick with the default video format. This also shows that users are possibly not interested in this feature or they are unaware of it. For Mobile-player the choice of resolution is either hidden or not available, and a marginal fraction of users exploit it.

Since resolution codes can be sorted, we can further classify the resolution changes in "Low-to-High" and "High-to-Low" considering the involved video resolution. Table 11.2 reports this breakdown. We can see that the Low-to-High resolution switch is largely predominant, with more than 80% (for all traces) being a 360p-Fl to 480p-Fl switch. Interestingly, when the full screen playback is enabled, the Flash player automatically switches from 360p-Fl to 480p-Fl (the converse is not

Data set	Low-to-High	High-to-Low
US-Campus	95.7	4.3
EU1-Campus	86.1	13.9
EU1-ADSL	93.9	6.1
EU1-FTTH	90.5	9.5
EU2-ADSL	83.6	16.4

Table 11.2. Percentage of resolution switch breakdown for PC-player in YT-2011.

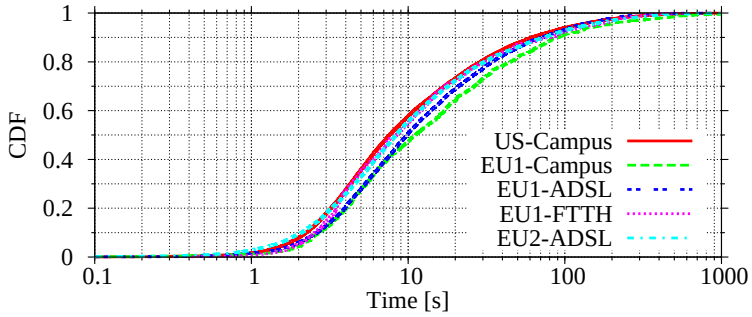


Figure 11.2. Time at which Low-to-High resolution switch happens in YT-2011.

true). Combining Tables 11.1 and 11.2, we can conjecture that full screen mode is not popular, but it is the main cause of resolution switch.

As final note, the largest majority of High-to-Low switch are 360p-Fl to 240p-Fl. This suggests that those are triggered by the user because of bad performance. EU2-ADSL and EU1-Campus show a slightly larger High-to-Low switch fraction. As we will see in Chapter 12 that these are the two vantage points with slightly worse performance.

To complete the analysis, we investigate when the resolution switch is triggered. Figure 11.2 shows the CDF of the time between the session start and Low-to-High resolution switch. Due to buffering at the player, this is an overestimate of the actual switch time. 50% of these events happens in the first 10 seconds, while only 10% of users trigger them after 1 minute. In terms of video size, more than 80% of the switches happens in the first 20% of the video data download while only 5% occurs in the second half of the video. The same consideration holds for High-to-Low changes. Overall we can conclude that resolution changes are usually performed at the very beginning of the playback.

We find surprising that results are practically identical in all data sets despite differences in users habits and cultures.

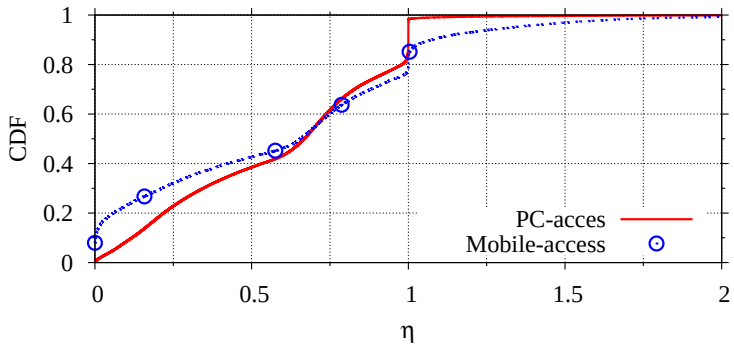


Figure 11.3. CDF of the fraction of downloaded video bytes. EU1-ADSL data set in YT-2011.

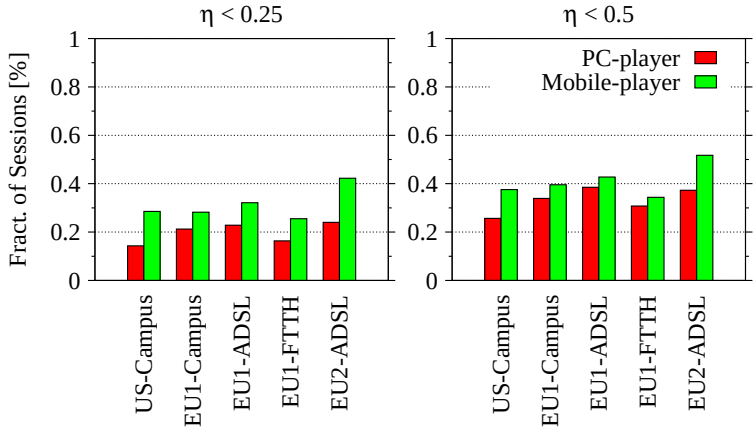


Figure 11.4. Fraction of video downloads having $\eta < 0.25$ and $\eta < 0.50$ in YT-2011.

11.3 Fraction of watched video

We now focus our attention on the time a user spends watching a video. To measure this, we leverage the fact that the player abruptly aborts the video download if the user changes the web page on the browser (or custom player). Let η be the fraction of downloaded video within a session with respect to the actual video size. If $\eta < 1$, then the user did not watch the entire video[1].

Figure 11.3 shows the CDF of η for EU1-ADSL data set. Two observations hold: i) about 80% of video sessions are abruptly interrupted; ii) Mobile-player results

[1] By checking the `Range` header field for requests, we filter out those sessions in which the user fast-forward the playback to a position outside the already buffered portion of the video.

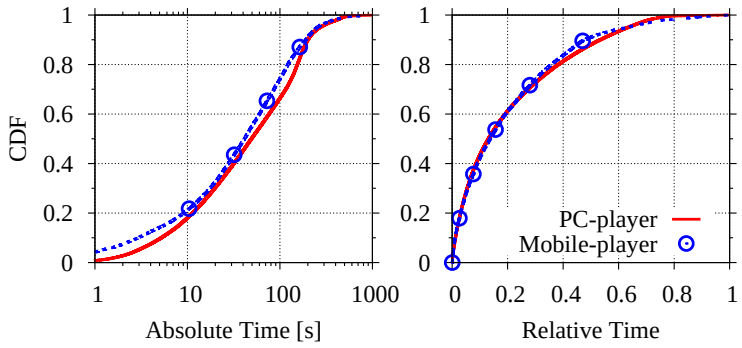


Figure 11.5. CDF of absolute (left) and relative (right) portions of watched videos for sessions with incomplete downloads. EU1-ADSL in YT-2011.

show that the player can download *more* data than the video size ($\eta > 1$). We will investigate this better next.

To better compare results, Figure 11.4 details the fraction of video downloads having $\eta < 0.25$ and $\eta < 0.5$ for different data sets. Interestingly, results are similar for all vantage points, with users on Mobile-player consistently aborting earlier than users on PC-player. Figure 11.5 details the absolute and relative time at which the user stops watching the video on the left and right plots, respectively. It shows that people tend to abort the playback very soon, with 60% of videos being watched for less than 20% of their duration. This can be due to a mismatch between the users' interests and the content they find on YouTube. Notice that this is also an interesting fact that could be exploited to better handle the content distribution among the CDN nodes, e.g., caching only some portion of each video. The impact of Mobile-player versus PC-player is very limited, testifying that the probability of aborting the playback is not biased by the device but it is related to users' habits.

11.3.1 Impact of buffering policy and user early abort

Consider now all video data already buffered at the player at the time the user aborts the playback. That data has been downloaded in vain. Figure 11.6 precisely quantifies this by reporting the ratio among downloaded bytes and the amount of bytes possibly consumed by the player. The latter is evaluated as $session\_duration * average\_encoding\_bitrate$, assuming that the playout started immediately after the first byte has been received, and that data is consumed at the video encoding bitrate. Since the initial buffering is neglected, our estimation can be considered as a lower bound. Notice also that we cannot evaluate the amount of wasted data for sessions which already have completed the download.

In spite of this, results are dramatic for PC-player: 40% of sessions download

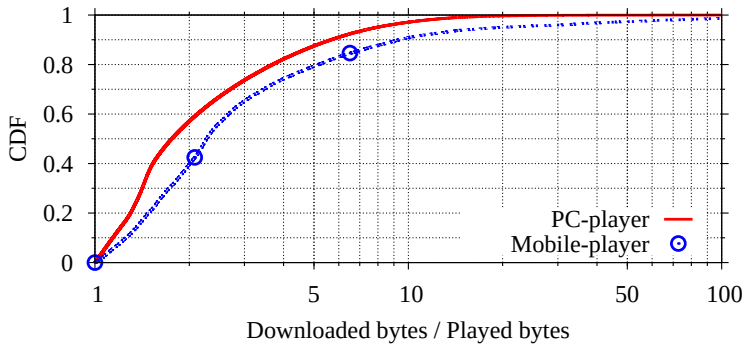


Figure 11.6. CDF of ratio between downloaded bytes and played bytes. EU1-ADSL in YT-2011.

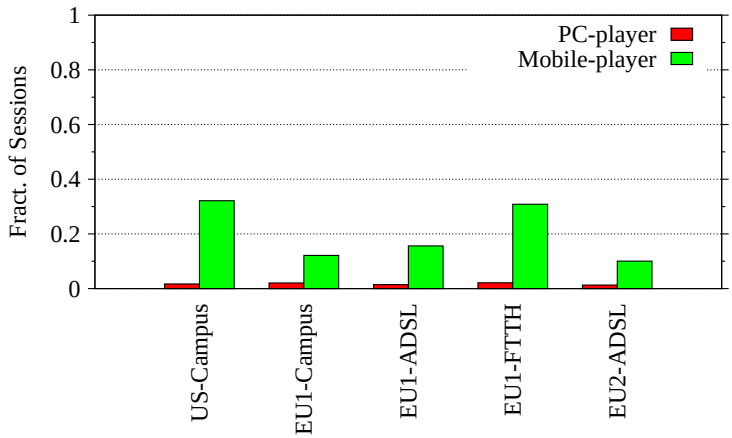


Figure 11.7. Fraction of sessions downloading more than the entire video in YT-2011.

more than two times the amount of data that was possibly watched. This is the result of aggressive buffering policies adopted by YouTube servers (recall that server-side shaping is adopted) [2]. Even worse, the Mobile-player waste is higher, with 20% of sessions downloading more the 5 times the amount of possibly watched data.

This waste could be reduced by implementing better streaming policies, e.g., the server sends chunks of the video by explicitly request by the client. Alternatively, a more accurate prediction of the fraction of the video that a user will watch can be leveraged to avoid transferring useless data.

Data set	PC-player	Mobile-player
US-Campus	39.17	47.9
EU1-Campus	36.91	38.1
EU1-ADSL	24.93	38.7
EU1-FTTH	38.43	53.5
EU2-ADSL	29.27	35.6

Table 11.3. Average percentage of wasted bytes considering peak hour with respect to useful data in YT-2011.

11.3.2 Sessions downloading more than the video size

Let us now focus on sessions where $\eta > 1$. Intuitively, those should be very limited, since one would expect that the player should not download more data than the total video size. Figure 11.7 shows the fraction of sessions for which this happens. Only sessions with no resolution switch are considered. For PC-player, less than 2% of sessions show this phenomenon. We have found that the exceeding amount of volume is possibly related to users watching the same video multiple times causing the player to re-download the video. Overall, this effect is marginal.

For Mobile-player instead we observe that 15-30% of sessions download more than the actual video size. Performing some active experiments, we have confirmed at least two causes for this: 1) in case of backward seeks, the player has to re-download the same content because it has been already discarded from the player's local buffer. This does not happen for PC-player which caches the entire video; 2) the aggressive chunk-based download mechanism is source of inefficiency: the Mobile-player often requests chunks bigger than needed, i.e., requesting from a desired position x up to the end of the video. The server then sends data from position x at a high rate, quickly filling up the Mobile-player buffer, e.g., up to position y. Being the buffer full, the player application abruptly closes the underlying TCP connection. However TCP had already received some data at the transport-layer receiver buffer up to position $y' > y$. The data $y' - y$ is thus discarded. When the application buffer depletes, the player requests data from position y and not from position y'. Considering the download of $y' - y$, the aggressive server buffering policy coupled with player limited buffering capabilities is thus origin of inefficiency.

To quantify the waste of traffic due to this, Figure 11.8 reports the CDF of the ratio of downloaded data versus nominal video size for sessions with $\eta > 1$. 50% of sessions download 25% more data, and 4% of the sessions downloads more than the twice of the video size.

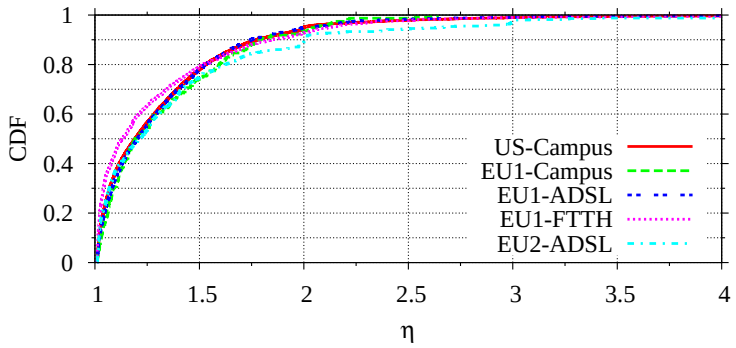


Figure 11.8. Ratio of downloaded data versus video size for sessions with $\eta > 1$. Mobile-player in YT-2011.

11.3.3 Wasted video data

Table 11.3 quantifies the overall percentage of wasted bytes with respect to useful data. It includes both the effect of aggressive buffer management and of chunk based video retrieval mechanisms. Measurements refer to the peak-hour time, when YouTube traffic peaks to several hundreds of Mb/s in most vantage points. Results show that the amount of traffic downloaded by clients but not used by players is comparable with the useful data traffic. For example, for US-Campus, the wasted traffic in a single hour amounts to 28.8 GB and 1.5 GB for PC-player and Mobile-player, respectively, corresponding to more than 67 Mb/s of constant waste. This is a large amount of wasted bandwidth both from the perspective of the ISP and the YouTube CDN.

We have performed experiments on mobile devices connected to a 3G network using both iOS and Android devices. The problem shows up exactly in the same way, with clients downloading a lot more data than the video played and the video size. We did not observe differences among different devices or operating systems. More recently, a study conducted in the AT&T cellular network [20] shows that this behaviour do not only affects YouTube but several others streaming systems. This suggests that the problem might be to a sort of "bug" in the playback framework and it would be critical given the increasing popularity of streaming services in 3G/4G networks.

11.4 Conclusion

Despite the rich set of video formats supported by YouTube, users are not interested in changing the resolution. Only 5% of the download from PC-player present the resolution switch mainly because of using the fullscreen feature, while this fraction

is negligible for Mobile-player downloads.

If one the one hand aggressive buffering policies are used to guarantee a continuity in the playback, on the other hand lead to unnecessary data transfer given that users watch just a portion of the video. This call for a future optimization of the system. First of all, more controlled streaming schemes can be used to i) limit the amount of data downloaded and not played back and ii) avoid the transmission of duplicated data for mobile devices. Moreover, CDN caching schemes can be improved by leveraging the fact that only a fraction of videos are actually watched by users.

Chapter 12

Streaming performance

In this chapter we focus on the YouTube streaming performance considering two metrics that reflect the user experience: *startup latency* and *bitrate ratio*. The first metric captures the initial delay experienced by the user before the playback starts. The second instead compare the speed of the download and the speed of the playback.

Both these indexes have been already presented in the literature [28, 67]. However, in our study we introduce more rigorous definitions and taking advantage of the YT-2011 data collection, we conducted a much deeper and complete characterization consistently comparing the performance with respect to the different type of devices used. Overall, the results show that, even if the download performance are good, the system is more optimized for PC-player downloads and there is space for future optimizations.

The remaining of this chapter is organized as follow: for each metric considered, we start presenting the definition and the measurements obtained. Then, we further drill into trying to assess the causes of performance impairments.

12.1 Startup latency

Recalling the description of the video download mechanisms reported in Chapter 7.2, to start the streaming the client performs a `videoplayback` request to a video server. We define the *startup latency* as the time elapsed between the `videoplayback` request and the first received packet containing actual video payload. This corresponds to a lower bound of the total delay experienced before the actual video playback starts since the initial buffering time at the player is ignored. Knowing the exact delay it is hard and implementation dependent. In the literature other definitions of the startup latency have been proposed measuring the time needed to retrieve the first MBs of the video [67]. However, we prefer to focus on a simpler, more precise

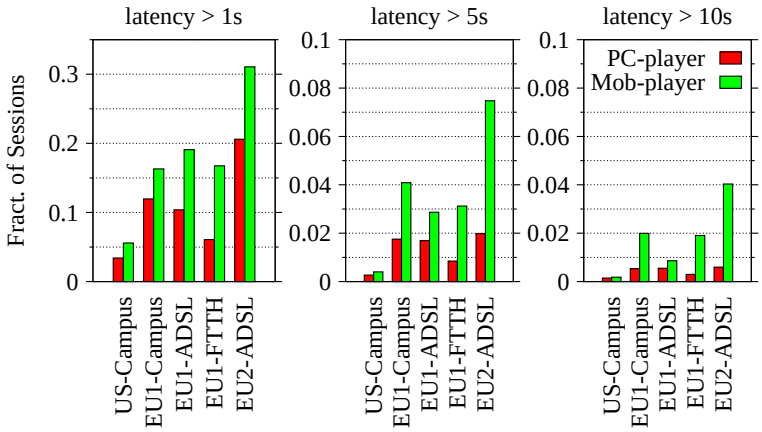


Figure 12.1. Fraction of sessions with high startup latency in YT-2011. Note the different y-axis across plots.

and accurate measure even if conservative.

Figure 12.1 reports the fraction of sessions with the startup latency higher than a certain threshold. Given that we are interested in studying the user experience, we selected threshold values that can be appreciated by the user, i.e., 1, 5 and 10 s. Results show that the performance is heterogeneous across the data sets, with Mobile-player suffering larger delays. For example, in the US-Campus less than 5% of sessions suffer a startup latency higher than 1 s. In the EU1-Campus instead, more than 10% of sessions start after 1 s with 2% of them suffering a startup latency higher than 5 s.

We found that the delay is due to a combination of causes.

Redirections: video sessions can suffer from a different number of redirections. Each redirection involves i) a DNS query to resolve the hostname of the next video server, ii) the opening of a new TCP connection, iii) a new video query. The network distance between the client and the server plays also a significant role, since YouTube CDN is likely to direct clients to video servers with the closest RTT. However, in case of redirections, the final server may not be the closest one in RTT (see Chapter 10).

Figure 12.2 reports the fraction of sessions affected by redirections. More than 70% of PC-player sessions do not suffer from redirections in all data sets, while Mobile-player sessions are more likely to be redirected. Understanding why this is happening is difficult. A possible cause of redirection is due to cache miss. In Chapter 10 we already showed that a cache miss at the closest data center, may cause a redirection to a farther away data center. However, following requests for the same content are directly served by the closest cache.

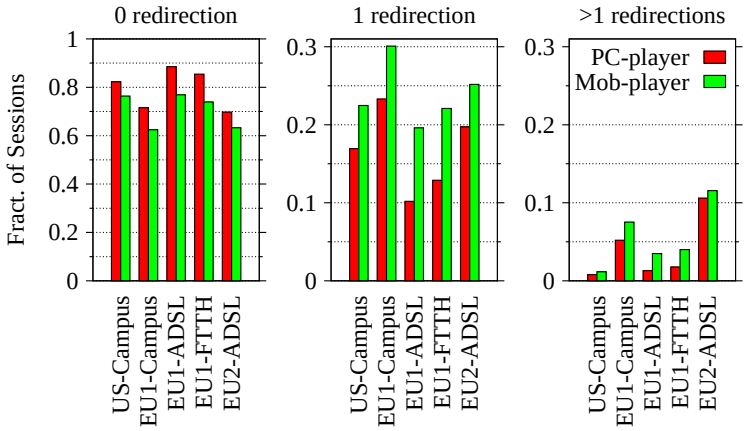


Figure 12.2. Fraction of sessions suffering redirections in YT-2011. Note the different y-axis across plots

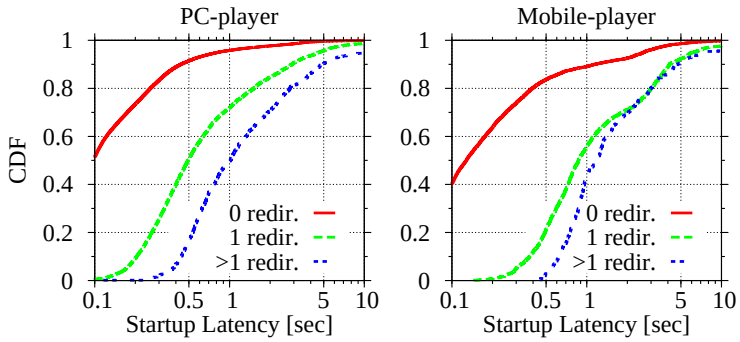


Figure 12.3. CDF of startup latency for different number of redirections. EU1-ADSL in YT-2011.

This hints to a caching mechanism based on pull schemes and it is confirmed by PlanetLab experiment reported in Chapter 10.4.3. Since Mobile-player videos are less frequently requested than PC-player videos because of the different video format adopted, a cache miss is possibly more likely to happen. Thus more redirections can occur.

Figure 12.3 depicts the impact of the redirections on the startup latency. We can see that the higher is the number of redirections in the video session, the higher is the startup delay. Considering PC-player (left plot), 92% of sessions that do not suffer redirections exhibit a startup latency smaller than 500 ms. When one redirection is faced, only 50% of sessions start within 500 ms. If

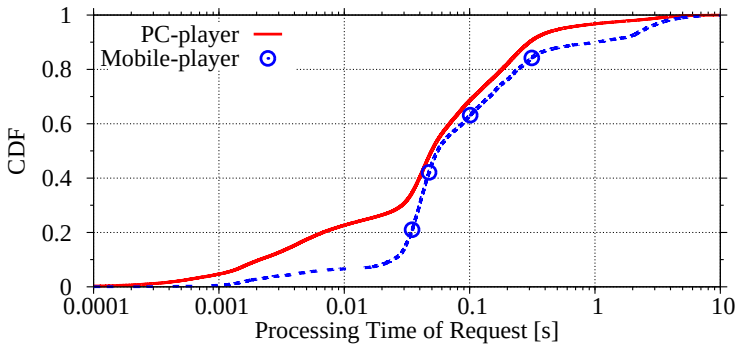


Figure 12.4. CDF of the video query processing time. EU1-ADSL in YT-2011.

more than 1 redirection is faced, more than 82% of sessions have a startup latency of at least 500 ms, with 10% of sessions suffering a latency higher than 5 s. Trends are similar for mobile player (right plot) where in general startup latencies are higher than in the case of PC-player.

Video request processing time: another possible cause of large startup time can be the server processing time, i.e., time needed by the video server to process a video request. To estimate it, we compute the time between the last `videoplayback` request sent by the client and the first video packet sent by the server. To eliminate the network delay we subtract the RTT[1]

Figure 12.4 reports the CDF of the estimated processing time for both PC-player and Mobile-player in the EU1-ADSL data set. Other data sets show similar trends. We can see that 50% of the requests are served within < 50 ms. A sharp knee around 30 ms is present and a heavy tail is found with processing time growing up to 5 s. The distribution reflects the time required by the cache to retrieve the requested content from the back-end before serving it. Very low latencies can be related to the video being already cached in the server memory; values in [30,300] ms can be related to disk access latency; finally values larger than 300 ms can be due to congestion in the back-end or to packet loss recovered by lengthy TCP timeout, or to rare content that has to be fetched from some slower storage system. The fact that Mobile-player responses require higher processing time can again be explained by the less frequent requests of Mobile-player video content. Note also that the prefetching mechanism implemented by PC-player could also speed-up the content retrieval (see Chapter 7.2).

[1]Tstat estimates the RTT by leveraging TCP acknowledgements [24, 72].

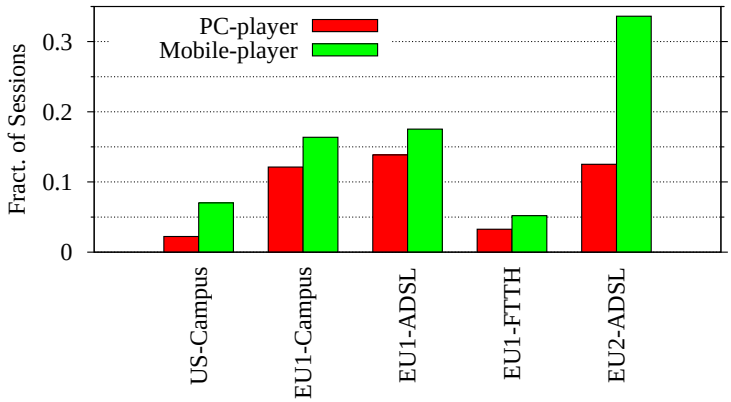


Figure 12.5. Fraction of sessions with bitrate ratio < 1 in YT-2011.

12.2 Bitrate ratio

The download bitrate of the video has a fundamental role in defining the quality of the video playback. In fact, if data is not received fast enough, buffer "under-run" events will be suffered, causing the video playback to pause. To measure the smoothness of the playback, we define the *bitrate ratio* as the ratio between the average session download bitrate and the video encoding bitrate. The first corresponds to the total amount of bytes downloaded aggregating flows of the same video session, divided by the time between the first and the last video packet. According to this definition, a bitrate ratio smaller than 1 is a clear sign of impaired performance.

Figure 12.5 reports the fraction of sessions with a bitrate ratio lower than one. Some interesting observations hold: first, the access technology has a clear impact on the performance with the ADSL networks performing worst for more than 10% of the downloads with respect to the other networks. For instance, compare EU1-ADSL and EU1-FTTH (the latter offers 10Mb/s full duplex access capacity). Both refer to customers of the same ISP in the same city. Still, EU1-ADSL customers suffer worse performance. Unexpectedly, EU1-Campus performs also quite bad. Further investigation revealed that this is the result of a local University network policy that limits the bandwidth of subnets of some dorms. Most of the sessions having poor performance are indeed coming from those subnets. Figure 12.5 shows that Mobile-player presents consistently lower performance than PC-player. This can be due to the presence of a WiFi network that is used by Mobile-player devices. The shared WiFi connection can indeed impair the download throughput. This is the case for US-Campus.

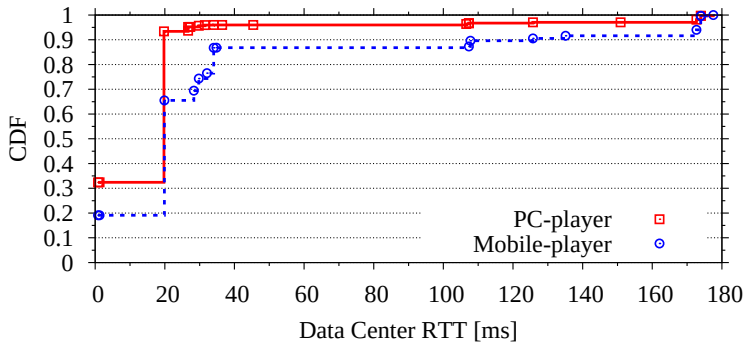


Figure 12.6. CDF of the fraction of bytes delivered by each data center. EU2-ADSL in YT-2011.

Other causes of reduced performance can be related to the YouTube infrastructure performing worst when serving Mobile-player requests. Consider EU2-ADSL, in which more than 32% of Mobile-player sessions are performing poorly versus less than 13% of PC-player sessions. We pinpoint that Mobile-player impaired performance is related to the YouTube system. Consider Figure 12.6. It reports the CDF of the fraction of bytes downloaded by different video servers respect to the RTT to the EU2-ADSL vantage point. Each point in the figure aggregates video servers that belong to the same CDN data center as seen in Chapter 10. We found that EU2-ADSL clients can use a data center which is very close to the vantage point (RTT $<$ 1 ms). However, it can only serve 35% of the PC-player sessions. The majority of sessions are indeed served by a second data center which is 20 ms far from the vantage point. For PC-player, these two data centers handle 96% of video requests. However, due to the lower popularity of Mobile player accessed content, 35% of Mobile-player sessions are served by other data centers, 10% of which are found outside Europe and suffer RTT $>$ 100 ms. These sessions can be impaired by network congestion and can exhibit lower download bitrate. Finally, recall the Mobile-player chunking mechanism. The cost of opening a new TCP connection to request a new chunk becomes significant when the RTT is in the order of hundreds of ms, impairing the download bitrate too.

In Chapter 11.2 we have seen that EU1-ADSL and EU2-ADSL presented an higher number of High-to-Low resolution switches. Interestingly, both these data sets present also lower download performance. This suggest that the users react to the performance impairments reducing the video resolution.

12.3 Conclusion

The measurements presented in this chapter show that Mobile-player performance result generally less efficient than PC-player. We think that this is the result of a design choice motivated by the intrinsically smaller popularity of Mobile-player accessed videos. Anyway, several studies forecast a strong increase in the volume of traffic for mobile devices [52, 49, 70]. This call for a better optimization of the resources, posing additional challenges to the YouTube CDN infrastructure.

In particular, we can speculate on different caching schemes that try to better address the caching of the videos with respect to the different format requested. If on the one hand Flash content represents the common type of video in the caches, on the other hand mobile devices prefer Mpeg content. Finding the proper balance between the two needs is fundamental. Moreover, other elements as the video popularity might be important as well defining better caching algorithms. Unfortunately, YT-2010 and YT-2011 data collections do not allows to measure these effects so we leave these analysis as future work.

Bibliography

[1] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. YouTube Traffic Dynamics and its Interplay with a Tier-1 ISP: an ISP Perspective. In *IMC '10*, 2010.

[2] Shane Alcock and Richard Nelson. Application Flow Control in YouTube Video Streams. *SIGCOMM Computer Communication Review*, 41:24–30, April 2011.

[3] M. Baldi, A. Baldini, N. Cascarano, and F. Risso. Service-based traffic classification: Principles and validation. In *SARNOFF '09*, 2009.

[4] P. Berkhin. A Survey of Clustering Data Mining Techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

[5] Paola Bermolen, Marco Mellia, Michela Meo, Dario Rossi, and Silvio Valenti. Abacus: Accurate Behavioral Classification of P2P-TV Traffic. *Computer Networks*, 55(6):1394–1411, 2011.

[6] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early Application Identification. In *CoNEXT '06*, 2006.

[7] Andrea Bianco, Gianluca Mardente, Marco Mellia, Maurizio Munafò, and Luca Muscariello. Web User-Session Inference by Means of Clustering Techniques. *IEEE/ACM Transactions on Networing*, 17:405–416, 2009.

[8] Joan-Isaac Biel and Daniel Gatica-Perez. Wearing a YouTube Hat: Directors, Comedians, Gurus, and User Aggregated Behavior. In *MM '09*, 2009.

[9] Robert Birke, Marco Mellia, Michele Petracca, and Dario Rossi. Understanding VoIP from Backbone Measurements. In *INFOCOM '07*, 2007.

[10] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype Traffic: When Randomness Plays With You. In *SIGCOMM '07*, 2007.

[11] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I Tube, You Tube, Everybody Tubes: Analyzing The World's Largest User Generated Content Video System. In *IMC '07*, 2007.

[12] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and Social Network of YouTube Videos. In *IWQoS '08*, 2008.

[13] Delia Ciullo, M.-A. Garcia da Rocha Neta, Ákos Horváth, Emilio Leonardi, Marco Mellia, Dario Rossi, Miklós Telek, and Paolo Veglia. Network Awareness

of P2P Live Streaming Applications: A Measurement Study. *IEEE Transactions on Multimedia*, 12(1):54–63, 2010.

[14] Classification Tool Suite. `http://netgroup.polito.it/research-projects/traffic-classification`.

[15] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, 2000.

[16] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic Classification through Simple Statistical Fingerprinting. *SIGCOMM Computer Communication Review*, 37:5–16, 2007.

[17] Christian Dewes, Arne Wichmann, and Anja Feldmann. An Analysis of Internet Chat Systems. In *IMC 03*, 2003.

[18] Endace DAG Home Page. `http://www.endace.com`.

[19] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic Classification Using Clustering Algorithms. In *MineNet '06*, 2006.

[20] Jeffrey Erman, Alexandre Gerber, K. K. Ramadrishnan, Subhabrata Sen, and Oliver Spatscheck. Over the Top Video: the Gorilla in Cellular Networks. In *IMC '11*, 2011.

[21] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/Realtime Traffic Classification Using Semi-Supervised Learning. *Performance Evaluation*, 64:1194–1213, October 2007.

[22] Jeffrey Erman, Anirban Mahanti, and Martin F. Arlitt. Internet Traffic Identification using Machine Learning. In *Globecom '06*, 2006.

[23] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support Vector Machines for TCP Traffic Classification. *Computer Networks*, 53:2476–2490, 2009.

[24] Alessandro Finamore, Marco Mellia, Michela Meo, Maurizio M. Munafò, and Dario Rossi. Experiences of Internet Traffic Monitoring with Tstat. *IEEE Network*, 25(3):8–14, 2011.

[25] Alessandro Finamore, Marco Mellia, Michela Meo, and Dario Rossi. KISS: Stochastic Packet Inspection Classifier for UDP Traffic. *IEEE/ACM Transactions on Networking*, 18(5):1505–1515, 2010.

[26] Tom Z. Fu, Yan Hu, Xingang Shi, Dah Ming Chiu, and John C. Lui. PBS: Periodic Behavioral Spectrum of P2P Applications. In *PAM '09*, 2009.

[27] Aaron Gember, Ashok Anand, and Aditya Akella. A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks. In *PAM '11*, 2011.

[28] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. YouTube Traffic Characterization: A View From The Edge. In *IMC '07*, 2007.

[29] F. Gringoli, Luca Salgarelli, M. Dusi, N. Cascarano, F. Risso, and k. c. claffy. Gt: Picking up the truth from the ground for internet traffic. *SIGCOMM Computer Communication Review*, 39:12–18, 2009.

[30] Gt Home Page. `http://www.ing.unibs.it/ntw/tools/gt/`.

[31] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based Geolocation of Internet Hosts. *IEEE/ACM Transactions on Networking*, 14(6):1219–1232, 2006.

[32] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. ACAS: Automated Construction of Application Signatures. In *MineNet '05*, 2005.

[33] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network Monitoring Using Traffic Dispersion Graphs (TDGS). In *IMC '07*, 2007.

[34] Ipp2p Home Page. `http://www.ipp2p.org/`.

[35] Yu Jin, Nick Duffield, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. Inferring Applications at the Network Layer Using Collective Traffic Statistics. In *SIGMETRICS '10*, 2010.

[36] Thomas Karagiannis, Andre Broido, Nevil Brownlee, kc claffy, and Michalis Faloutsos. Is P2P Dying or Just Hiding? In *Globecom '04*, 2004.

[37] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport Layer Identification of P2P Traffic. In *IMC '04*, 2004.

[38] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *SIGCOMM '05*, 2005.

[39] Thomas Karagiannis, Konstantina Papagiannaki, Nina Taft, and Michalis Faloutsos. Profiling the End Host. In *PAM '07*, 2007.

[40] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *CoNEXT '08*, 2008.

[41] Ron Kohavi and Ross Quinlan. Decision Tree Discovery. In *In Handbook of Data Mining and Knowledge Discovery*, 1999.

[42] Yoram Kulbak and Danny Bickson. *The eMule Protocol Specification*, 2005.

[43] l7filter Howto. `http://l7-filter.sourceforge.net/HOWTO`.

[44] Gianluca La Mantia. Statistical Classification of TCP Traffic with Support Vector Machines. 2008.

[45] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet Inter-Domain Traffic. In *SIGCOMM '10*, 2010.

[46] E. Leonardi, M. Mellia, A. Horvath, L. Muscariello, S. Niccolini, and D. Rossi. Building a Cooperative P2P-TV Application Over a Wise Network: the Approach of the European FP-7 Strep NAPA-WINE. *Communications Magazine, IEEE*, 46(4):20–22, 2008.

[47] Libsvm Home Page. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[48] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected Means of Protocol Inference. In *IMC '06*, 2006.

[49] Gregor Maier, Fabian Schneider, and Anja Feldmann. A First Look at Mobile Hand-Held Device Traffic. In *PAM '10*, 2010.

[50] Maxmind GeoLite City Database. `http://www.maxmind.com/app/geolitecity`.

[51] Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM '04*, 2004.

[52] The Mobile Internet Report. `http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html`.

[53] Andrew W. Moore and Konstantina Papagiannaki. Toward the Accurate Identification of Network Applications. In *PAM '05*, 2005.

[54] Andrew W. Moore and Denis Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *SIGMETRICS '05*, 2005.

[55] David Moore, Ken Keys, Ryan Koga, Edouard Lagache, and k claffy. The CoralReef Software Suite as a Tool for System and Network Administrators. In *Usenix LISA '01*, pages 4–7, 2001.

[56] Narus. `http://www.narus.com`.

[57] ntop Home Page. `http://www.ntop.org`.

[58] Opendpi Home Page. `http://www.opendpi.org`.

[59] Vern Paxson. Bro: a system for detecting network intruders in real-time. In *USENIX SSYM '98*, 1998.

[60] Marcin Pietrzyk, Jean-Laurent Costeux, Guillaume Urvoy-Keller, and Taoufik En-Najjary. Challenging Statistical Classification for Operational Usage: the ADSL Case. In *IMC '09*, 2009.

[61] R. Percacci and A. Vespignani. Scale-free Behavior of the Internet Global Performance. *European Physical Journal B*, 32(4):411–414, 2003.

[62] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network Characteristics of Video Streaming Traffic. In *CoNEXT '11*, 2011.

[63] F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus. Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation. In *ICC '08*, 2008.

[64] 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 *IMC '04*, 2004.

[65] Sandvine Home Page. `http://www.sandvine.org`.

[66] Mohit Saxena, Umang Sharan, and Sonia Fahmy. Analyzing Video Services in Web 2.0: a Global Perspective. In *NOSSDAV '08*, 2008.

[67] Mohit Saxena, Umang Sharan, and Sonia Fahmy. Analyzing Video Services in Web 2.0: a Global Perspective. In *NOSSDAV '08*, 2008.

[68] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13:1443–1471, 2001.

[69] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. In *WWW '04*, 2004.

[70] M. Zubair Shafiq, Lusheng Ji, Alex X. Liu, and Jia Wang. Characterizing and Modeling Internet Traffic Dynamics of Cellular Devices. In *SIGMETRICS '11*, 2011.

[71] Skype Testbed Traces. `http://tstat.tlc.polito.it/traces-skype.shtml`.

[72] Tstat Home Page. `http://tstat.polito.it`.

[73] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *SIGCOMM '05*, 2005.

[74] Youtube API. `http://code.google.com/apis/youtube/overview.html`.

[75] YouTube Blog: Mmm mmm good - YouTube videos now served in WebM. `http://youtube-global.blogspot.com/2011/04/mmm-mmm-good-youtube-videos-now-served.html`.

[76] YouTube Blog: Speed Thrills - Tackling the YouTube video processing challenge. `http://youtube-global.blogspot.com/2011/02/speed-thrills-tackling-youtube-video.html`.

[77] YouTube Press Room. `www.youtube.com/t/press_statistics`.

[78] Min Zhang, Maurizio Dusi, Wolfgang John, and Changjia Chen. Analysis of UDP Traffic Usage on Internet Backbone Links. In *SAINT '09*, 2009.

[79] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Computer Networks*, 53(4):501–514, 2009.