Programme ANR VERSO

Projet VIPEER

Ingénierie du trafic vidéo en intradomaine basée sur les paradigmes du Pair à Pair

Décision nº 2009 VERSO 014 01 à 06 du 22 décembre 2009
T0 administratif = 15 Novembre 2009
T0 technique = 1er Janvier 2010

Livrable 2.2

Technical specifications of the modules proposed within the WP2: Traffic Classifier, QoE Evaluater and their integration

Auteurs:
Y.Hadjadj-Aoul, K.Singh (INRIA), S.Vaton, M.K.Sbai (Telecom Bretagne), S.Moteau (Orange lab.)
Edited by:
Y.Hadjadj-Aoul (INRIA)

Décembre 2011

INRIA, Telecom Bretagne; Orange lab.

Abstract

Developing a distributed CDN (dCDN) in the intra-domain network, which is the main objective of the ViPeer project, clearly requires the development of some tools and algorithms in order to improve the performance of such system by reacting to network parameters variations. Thus, different tools and algorithms are developed within the working group 2 of the ViPeer project. These tools will not only help in monitoring the network but also can be used for some control actions aimed to improve the global performance of the proposed dCDN. The present deliverable presents, particularly, the technical specifications of the first versions of traffic classifier and of the QoE evaluation module.

Keywords: network monitoring, QoE, QoS, traffic classification, testbed

Contents

1 Introduction 5

2 Technical specifications of the Traffic Classifier 7
2.1 Introduction . . . 7
2.2 A Support Vector Machine based classifier . . . 9
2.2.1 Background on SVM . . . 9
2.2.2 Accuracy of the SVM classifier . . . 11
2.3 Implementation of the SVM classifier . . . 13
2.3.1 Motivations . . . 13
2.3.2 Centralized architecture of the classifier . . . 14
2.3.3 Boosting SVM performance with NetFPGA hardware implementation . . . 15
2.4 Implementation of the SVM learning phase . . . 16
2.4.1 Stability of SVM to learning trace . . . 16
2.4.2 A community-based SVM learning . . . 18
2.4.3 Accelerating SVM Learning using GPU . . . 20
2.5 Conclusion . . . 20

3 Technical specifications of the QoE Evaluater 23
3.1 Introduction . . . 23
3.2 QoE Estimation . . . 24
3.2.1 Pseudo-Subjective Quality Assessment (PSQA) . . . 24
3.2.2 Adaptive HTTP Streaming . . . 24
3.2.3 Input parameters for QoE evaluator . . . 25
3.3 QoE Evaluator . . . 26
3.4 Conclusion . . . 28

4 Tools' Integration within the testbed 31
4.1 Architecture reminder . . . 31
4.2 Communication between the different entities . . . 32
4.3 Global architecture . . . 35
4.4 Conclusion . . . 36

5 Conclusions 37

Bibliography 39

Bibliography 38

1 Introduction

This deliverable provides the technical specifications for the different integral tools of the global measurement infrastructure. The objective of the measurement infrastructure tools is to inform the network and QoS aware dCDN with useful information about the network state and about the end user activity. These measurements, along with QoE estimation, will not only serve as a global indicator of network performance and customer satisfaction, but also in making long term decisions, such as the network planning. Meanwhile, this monitoring could also help in determining both the optimal storing strategy and the best reaction to services' quality degradation. The ultimate goal is to be able to optimize the distribution strategy of the dCDN so as to minimize the impact of the dCDN on the QoS perceived by the set-top-box owners and to maximize the QoS experienced by the dCDN's users.

The functionalities provided by the measurement infrastructure are manifold. The tools should be able to monitor the network parameters such as available bandwidth, packet loss, delay and jitter on the access links: in particular the uplinks of the dCDN elements that provide caching and content distribution. Moreover, it is important to know the type of applications running on the peers. As similar network performance parameters can impact QoE differently for different applications, this knowledge about the available up/downlink bandwidth in addition to other network performance parameters as well as the type of applications running on a peer can help in two ways. First it can help in mapping the network parameters to application specific QoE. Second it can help in choosing the optimal peers that can devote a chunk of their available bandwidth without degrading their QoE. Lastly, the tools should be able to automatically estimate of the Quality of Experience that is delivered to the different end-user applications, taking into account the network state in real time. This can be used for admission control and congestion control when QoE gets degraded.

Measurement of network performance parameters is provided by state of the art techniques such as packet pair and other techniques that can be classified as either active or passive. The tools used for traffic classification and QoE estimation are based on supervised learning techniques. Traffic classification can be done using Deep Packet Inspection (DPI), but it is extremely demanding on high bandwith links and cannot be used if the applications cipher their traffic. Thus, a classifier based on Support vector machine (SVM) is used that employs statistical analysis of some traffic descriptors such as packet or flow-level characteristics of first three data packets in order to classify flows. For QoE estimation a tool called PSQA (pseudo subjective quality analysis) based on random neural networks (RNN) is used.

More details about these tools are provided in the following chapters. Chapter

2 describes the traffic classification tool. Chapter 3 focus on QoE estimation and Chapter 4 talks about the integration of all the tools.

2 Technical specifications of the Traffic Classifier

2.1 Introduction

In the monitoring architecture described in deliverable D1.2 we have suggested using traffic composition as a useful metrics both for selecting nodes to whom one should push video chunks and from which one should retrieve them. Indeed, using a portion of the upload/download capacity of nodes for P2P video delivery and storage can impact the QoS perceived by other applications that are running on a particular host. Consequently, the available bandwith is not a sufficient metrics to make an optimal selection of video streaming peers. One would like to have a deeper view on the activity of nodes, in particular to know which categories of applications are running on them. As a consequence it is useful to design an architecture that allows traffic classification.

Traffic classification is the task of associating network traffic with the application or category of applications that has generated it. Traditional methods for traffic classification are based (i) either on the analysis of port numbers (ii) or on the analysis of the application layer payload. Port numbers and payload analysis can be used in many cases in order to identify applications. But it is well known that these methods are not any longer fully reliable. Some applications use dynamic port numbers or do some tunneling in order to "hide" themselves behind other applications. Deep Packet Inspection (DPI, that is to say payload based filtering) is extremely demanding on high bandwith links and cannot be used if the applications cipher their traffic. Consequently some methods have been designed in order to identify applications without port numbers or payload inspection. These methods are based on a statistical analysis of some traffic descriptors such as packet or flow-level characteristics (size, timestamps).

Leveraging some previous work published in [17] the approach adopted in deliverable D2.1 uses the sequence of TCP flags as traffic descriptors. More precisely the TCP flags of the first packets of a TCP connection are used as inputs by a decision making algorithm (Neyman-Pearson test, Maximum Likelihood criterion, ...) This approach has been implemented in a software tool, so-called Nicofix in deliverable D2.1., developed at Telecom Bretagne in the framework of some projects with our students.

Although the first results were promising the method has not been considered good enough in the general case. Indeed the dataset used for the validation in

[17] is not fully representative of Internet traffic. It is based on a measurement campaign over an Orange GPRS network in 2006. Internet traffic over GPRS is not as heterogeneous as ADSL traffic and this has skewed the validation of the method. One of the challenges that lightweight traffic classification methods have to face is the diversity of applications it should be able to classify. On the contrary in a GPRS traffic trace dating back to year 2006 there are very few applications and this facilitates the task of any classifier. The method has other limitations such as its inability to deal with other traffic than TCP traffic whereas many interesting applications are using UDP as transport protocol.

Consequently other methods have been considered for integration into the software tool that we develop. SVM (Support Vector Machine) is often considered as the best performing algorithm for traffic classification [18] [19][20]. We have implemented a classifier which uses the first three data packets of a flow as traffic descriptor and a Support Vector Machine in order to classify flows. We have checked by extensive validation over different datasets with groundtruth identified that the performance of the method in terms of its accuracy (i.e. classification rate) is good as it was reported in the litterature.

The performance of different classification techniques has been deeply investigated in terms of the obtained classification rates (i.e. % of flows which are correctly associated to the generating application) . But in spite of the plethora of litterature about traffic classification the question of how these methods must be implemented *in practice* to enable online traffic analysis has not received enough attention. There are a few studies that investigate the impact, for example, of packet or flow level subsampling [21] or feature selection [22] on classification accuracy. But there is a lack of litterature about boosting lightweight traffic classification algorithms with hardware and/or software acceleration techniques.

In this chapter, we adress the design of a classifier based on Support Vector Machines. First, we validate the accuracy of the SVM classifier for different datasets. We first demonstrate that when the same dataset is considered both for the learning and the detection phases, the accuracy of the SVM algorithm is good. Then we study the stability of the SVM algorithm and its sensitivity to the learning trace and demonstrate that extreme care should be dedicated to the design of the learning phase. The learning phase is the calibration of traffic models (e.g. Support Vector Machines) by analysis of traffic datasets with groundtruth established. These models are afterwards used in order to classify new flows. We demonstrate by simulations over different datasets that the composition of the learning trace has an impact on the performance of the SVM algorithm. Then, we describe a community-based learning architecture that allows building learning traces that are representative of the network where the classification algorithm is running.

Another contribution concerns software and hardware acceleration for improving the performance of both the learning phase and the detection phase. The training phase of SVM can be in some cases an extremely time consuming task. In order to speed up the learning phase we compare different software acceleration approaches: parallelization with OpenMP on a multi-core CPU architecture and massive parallelization on the GPU (Graphical Processing Unit). The training phase of SVM classification is usually performed offline and consequently there is no real time con-

straint for the training phase contrary to the classification phase. But as the volumes of traffic that should be treated in order to calibrate the models can be huge it is necessary to leverage on the recent evolutions in the field of high performance computing (HPC) in order to make the most of up to date technologies. Traffic classification in itself must be performed on line in many cases in order to permit a fast adaptation of the system to the results of the traffic analysis. Depending on the point where the probe is collected a hardware accelerated classifier can be necessary. In order to investigate the possibility of designing hardware accelerated versions of SVM based traffic classifiers we have used NetFPGA 1G boards. NetFPGA is a project of Stanford university with sponsoring of Xilinx among others. NetFPGA boards are FPGA hardware boards dedicated to research and teaching in the field of traffic processing. Two versions of boards exist: NetFPGA 1G boards with 1 Gb/sec interfaces and the new NetFPGA 10G boards with 10 Gb/sec interfaces. The results that we have obtained are promising.

2.2 A Support Vector Machine based classifier

2.2.1 Background on SVM

Support Vector Machine [13] is a supervised classification algorithm. SVM transforms a non linear classification problem into a linear one, using a so called "kernel trick". Given a set of sample points in a multi-dimensional space one would like to separate them by hyperplanes, thus defining different classes. In many cases it is impossible to separate sample points of different classes by hyperplanes and the separating surface is extremely difficult to compute. The idea of SVM is to map, by means of the kernel function, training points to a transformed space of higher dimensionality where it is possible to find separating hyperplanes. In the target space SVM must find the hyperplanes which separate points belonging to different classes and have a maximum distance from misclassified points of both classes to the separating hyperplanes. The output of the training phase is made up of the parameters of the kernel and a set of support vectors that define the separating hyperplanes. During the classification phase SVM simply classifies new points according to the subspace they belong to.

SVM is often regarded as the best performing algorithm for traffic classification [23][22] and has been adopted by several authors [24][16][15]. The accuracy depends on the selection of the kernel functions where Gaussian kernels usually give good results. We use in our implementations the LibSVM [25] library, which is an integrated software for support vector classification allowing multiclass classification, learning, cross-validation and different kernel functions.

LibSVM implements different algorithms for applications of SVM to classification, to distribution estimation and to regression problems. There exist several algorithms for SVM based classification. We have used the C-Support Vector Classification (C-SVC) algorithm [26] that is described below.

Let us assume that we have a set of training points $x_i \in \mathbb{R}^n, i = 1, \ldots, l$ in two classes and a set of indicator values $y_i \in \{-1, +1\}$ such that $y_i = +1$ if x_i belongs to

class 1 and $y_i = -1$ if x_i belongs to class 2. Let us also assume that we have selected a function ϕ such that $\phi(x_i)$ maps training point x_i into a higher dimensional space. The search of an hyperplane that separates points $\phi(x_i)$ belonging to classes 1 and 2 comes up to solving the following optimization problem:

$$\begin{array}{ll} \min_{w,b,\zeta} & \frac{1}{2}w^T w + C\sum_{i=1}^{l}\zeta_i \\ \text{subject to} & y_i(w^T\phi(x_i)+b) \geq 1-\zeta_i \\ & \zeta_i \geq 0, i = 1,\ldots,l \end{array} \tag{2.1}$$

In the above equation the vector w defines the direction of the separating hyperplane and $\zeta_i, i = 1,\ldots,l$ are slack variables. C is a regularization parameter that penalizes solutions where some points are misclassified and or close to the separating hyperplane.

w is a vector in a high-dimensional space. In order to reduce the dimension of the optimization problem it is usual to consider the following dual problem:

$$\begin{array}{ll} \min_{\alpha} & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} & y^T\alpha = 0 \\ & 0 \leq \alpha_i \leq C, i = 1,\ldots,l \end{array} \tag{2.2}$$

where $e = [1,1,\ldots,1]^T$, Q is an l by l positive semidefinite matrix with elements $Q_{ij} = y_i y_j K(x_i,x_j)$ and $K(x_i,x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel function.

The solution of the dual problem is a vector α. Once the dual problem is solved the optimal w is given by:

$$w = \sum_{i=1}^{l} y_i\alpha_i\phi(x_i) \tag{2.3}$$

In the classification phase any new point x is classified according to the following decision function:

$$\text{sign}(w^T\phi(x)+b) = \text{sign}(\sum_{i=1}^{l} y_i\alpha_i K(x_i,x)+b) \tag{2.4}$$

The equation of the separating hyperplane is given by: $w^\phi(x)+b = 0$. x is classified into class 1 if $w^\phi(x)+b$ is positive and into class 2 if $w^\phi(x)+b$ is negative.

In what follows we use a Gaussian kernel as good results are often obtained when selecting this kernel. The equation of the Gaussian kernel is given by:

$$K(x_i,x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2) \tag{2.5}$$

The training phase of C-SVC produces as an output the equation of the separating hyperplane and the corresponding decision function.

From this simple two-classes SVM problem, one can deal with multi-class SVM classification problems. A usual approach is the so called "one versus one" (1 vs 1) approach. In the 1 vs 1 approach $n(n-1)$ two-classes SVM problem are considered, one for each pair of classes. A training phase is performed for each two-classes SVM problem thus producing n(n-1) separating hyperplanes . Each new point is then classified according to each of those two-class classification problems. The final

decision is taken on the basis of a majority vote, that is to say that the new point is allocated to the class which has obtained the highest number of points.

As one can see from the above description from the above description of the training phase (Equations 2.2, 2.3, 2.4, 2.5) other parameters need to be tuned , namely the regularization parameter C and the parameters of the kernel function (γ in the case of a Gaussian kernel). Cross-validation aims at tuning those parameters in order to fully calibrate the classifier. The principle of cross-validation is to test the accuracy obtained with many different values of the parameters C and γ and to keep the best performing selection. The state space $\mathbb{R}^2$ of parameters (C, γ) is sampled thus obtaining a grid. For each point (C, γ) of this grid the separating hyperplanes are obtained as a solution of the training phases and the percentage of training points which are accurately classified is computed. One keeps as an output of the cross-validation the best performing value of parameters (C, γ).

2.2.2 Accuracy of the SVM classifier

In order to assess the accuracy of the SVM based classifier we have performed validation over three different datasets. The learning and detection phases have been performed using the libSVM library [**?**]. The traffic descriptor that is used as input to the SVM classifier is made up of the size of the first three non empty packets of each flow, where a flow is defined as a set of packets with identical 5-tuples (IP Src adress, IP Dest adress, Src port, Dest port, protocol).

We have used for validation three datasets with groundtruth. The groundtruth identifies the application that has generated the traffic flow. It has been obtained either by Deep Packet Inspection (DPI) with for example Linux L7-filter [27] or by using a tool such as GT [28]. GT has been developed by the university of Brescia. GT is based on the analysis of logs of system calls generated by the different applications and their correlation with traffic dumped on one network interface of a host machine. GT also embeds DPI with L7-filter.

The names and characteristics of the three traffic traces used as benchmarks are listed in Table 2.1. Those three traces correspond to three very different scenarios: campus network, laboratory environment and residential access network. As a consequence the composition of traffic is significantly different from one trace to the other.

1. The FT (France Telecom) dataset has been provided by France Telecom under the terms of a Non Disclosure Agreement. Traffic has been dumped on one geographical zone of an ADSL France Telecom access network and groundtruth has been established by DPI.

2. The Ericsson dataset corresponds to some trafic that has been captured in a laboratory environment of Ericsson research.

3. The Brescia dataset is a public dataset [28]. It corresponds to some traffic captured on a campus network. The groundtruth has been obtained with the GT tool of the university of Brescia.

Trace label	Network of capture	Number of flows
FT	a DSL Link of France Telecom	304095
Ericsson	Local Area Network at an Ericsson Laboratory	12616
Brescia	Campus trace generated at University of Brescia, Italy	112337

Table 2.1: Traffic traces

The definition of classes is not universal. It mainly depends on the filters that have been defined for packet payload inspection (DPI). In order to enable a comparison between traces we have merged applications into different categories of applications. The different categories of applications are listed in Table 2.2.

Class label	Class name
1	Web
2	P2P download
3	Direct download
4	Streaming
5	Game
6	Mail
7	Instant messaging
8	Distant control

Table 2.2: Traffic classes

Table 2.3 provides figures of traffic classification accuracy for each of the three traces. In this Table the accuracy represents the overall percentage of flows which are correctly classified. We take into account all classes in this assessment of classification accuracy. It is worth noting that in this scenario the SVM model has been trained on the same dataset as the one used for classifying flows. As one can see from this table the performance of SVM based traffic classification is very good in this scenario.

A global accuracy figure is usually not considered as sufficient to demonstrate the performance of a classifier. Indeed some classes could be frequently misclassified with not much impact on the global accuracy figure if only few flows correspond to those classes. A usual representation of traffic classification results is given by the confusion matrix. In this report we provide in Figure 2.1 the accuracy per category of application, that is to say the percentage of flows of each category of applications that has been accurately classified.

As one can see from this Figure, the accuracy of the SVM algorithm differs from one category of applications to another and from one trace to another. The proportion of a category of applications in a trace impacts the ability of the SVM algorithm to detect it. For example, as class 1 (Web) is present with a good proportion in all

three traces, the accuracy of the detection is high. However, as class 4 (Streaming), is almost absent in the three traces it has the worst classification accuracy.

Trace	FT	Ericsson	Brescia
Accuracy (in %)	94.43	98.53	97.41

Table 2.3: Accuracy of the SVM algorithm

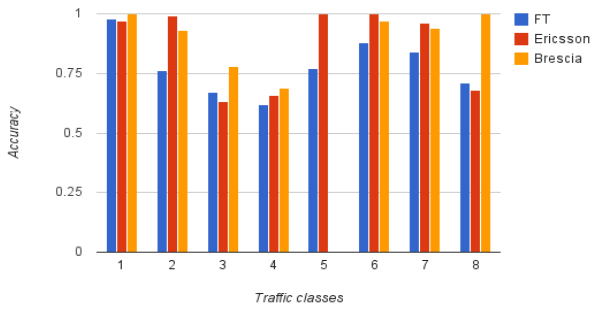

Figure 2.1: Accuracy per traffic class

2.3 Implementation of the SVM classifier

2.3.1 Motivations

Many factors must be taken into consideration when implementing the SVM classifier in a real network. In particular operators are very sensitive to the cost of deploying dedicated monitoring solutions. The benefits of monitoring should balance the cost (CAPEX and OPEX) of deploying and operating such a dedicated infrastructure. This is one of the reasons why the efforts of the academic community in the field of network monitoring have not conducted to a bunch of commercial products.

This pleads in favor of a solution that is based on commodity monitoring solutions such as NetFlow. Although the input of the above described SVM classifier is a flow level record such as NetFlow/IPFIX flow records the size of the first packets are not included in standardized versions of flow level records. This mitigates the possibility of deploying this classifier easily in an operated network. Despite this limitation it is likely that an operator would prefer collecting records with commodity flow level probes and analyzing the records at a central collector with enough computational power. Indeed if sophisticated traffic analysis algorithms are executed at the level of a central collector it is not necessary to deploy dedicated probes and it should be possible to leverage on the already deployed equipments. This is why the approach

adopted in VIPEER and described in deliverable D2.1 is in favor of a central collector analyzing the records of many probes. From another point of view, centralizing flow records and using one super-computer for the classification can consume important network bandwidth for the collection of records if the system is deployed on a large scale.

Another option is to implement a classifier for each end-point or group of end-points (subnetwork). This would require the deployment of probes with lightweight classification functionalities enabled and this deployment could be considered as overly costly as these are dedicated probes. Another challenge is that, depending where the probe is located, the data rate to be processed can be high. It is consequently interesting to investigate the possibility of deploying accelerated probes. The approach investigated in this section is to use NetFPGA boards in order to design hardware FPGA accelerated classifiers.

2.3.2 Centralized architecture of the classifier

To run the SVM classification algorithm for different capture points in the network, one needs to design a distributed architecture for the classifier. In this architecture as summarized in Figure 2.2, some detection clients capture traffic in the network, construct flow records, and send them to a centralized collector that stores them in a detection database. After that, a detection process runs the classification algorithm on them and stores the results in the same database. It is possible to visualize traffic composition through a web interface to that database. The software and data components orchestrated in the detection plane are:

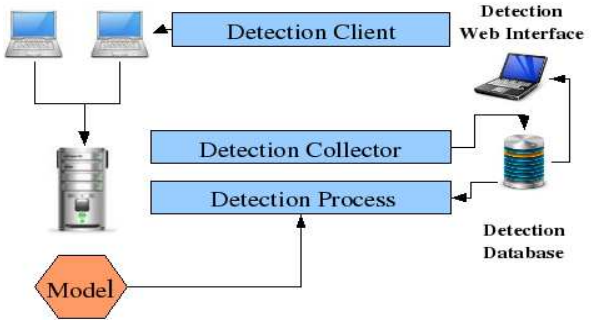

Figure 2.2: Architecture of the Detection plane

- Detection Client: This software component allows to capture traffic at any device in the network which is equipped with a DAG card. It constructs traffic flows online from the captured packets and sends information needed for the classification operation to a central detection collector. For example, the detection client sends for each flow the IP addresses, ports, the protocol and the size of first three data packets as these are the information required by the SVM detection algorithm described is Section 2.2.

- Detection Collector Module: This software component runs on a central server necessarily deployed by the organism that supports the classification architecture. It receives permanently flow records from different Detection Clients whose functionalities are described above. It inserts these flow information into the Detection Database.

- Detection Database: It is a database co-located with the Detection Collector. It contains information on the flows to be classified and the classification results of flows that have been detected. This database is mainly used and updated by the Detection Process. It is also consulted by the Detection Web Interface.

- Detection Process: This process runs on the same server as the Detection Database. It always read flows information added by the Detection Collector to the database. It uses the classification model provided by the Learning Process and the Detection algorithm to give labels to different flows. These classification results are written to the Detection Database.

- Detection Web Interface: This web interface allows users to connect to the Detection Database and view the classification results of their own traffic. Moreover, superusers can visualize more global results and understand the composition of the traffic transiting in the network.

This architecture has already been described in deliverable D2.1. A first implementation of the components of the architecture has been performed by Telecom Bretagne. A set of tests has been performed with the participation of France Telecom and Telecom Bretagne. In those tests the collector was located at France Telecom (in a laboratory in Lannion) and a probe was located in a laboratory at Telecom Bretagne. The detection clients at Telecom Bretagne sent flow records to the Detection Collector Module at France Telecom which stored them in a detection database and processed them by the detection algorithm. The architecture was functional but the series of tests have demonstrated the sensitivity of the classifier to the specification of traffic models which are the outputs of the learning phase. In section 2.4 we discuss into details a learning architecture which could enable the adaptation of traffic models to the traffic trends of the monitored network (releases of new versions of softwares, release of new applications, etc ...)

2.3.3 Boosting SVM performance with NetFPGA hardware implementation

As discussed in the introduction of this section another solution would be to design probes which are able to classify traffic using a lightweight classification method such as SVM. Those probes should be able to process traffic at wire speed if no subsampling is envisioned. It is well known that subsampling often skews traffic characteristics. As a consequence subsampling should be considered with extreme care when designing a lightweight classifier. For that reason we have decided to investigate the possibility of designing a lightweight traffic classifier without subsampling and able to process traffic at wire speed.

Even if the probe is located relatively close to end-users the data rate that must be processed is very high. As a matter of example the data rate of the Internet access

of Telecom Bretagne is close to 1 Gb/sec. An ordinary computer does not have the interfaces and the computational power to deal with such a traffic rate. To deal with high rate traffic, we are currently working on implementing the classification phase on a NetFPGA [29] board. As simple calculations can be highly parallelized on a FPGA, it should enable to process traffic on-line with a much higher bandwidth (the NetFPGA 10G has 4×10Gb/s interfaces). Our implementation is divided into two blocks:

- The first block is integrated into the NetFPGA platform: it receives packets and reconstitutes data about the size of the first packets of the flow.
- The second block is a configurable SVM classification block which receives a vector with the set of traffic features and returns the class it belongs to. The configuration of the block (SVM model) is stored in the RAM of the FPGA.

Our first results using a synthetic traffic generator and our implementation of the SVM detection algorithm on a NetFPGA 1G board are promising. In fact, we generate traffic with different types of application classes at a 1 Gbit/s rate. The NetFPGA implementation can classify all flows online without loosing any flow and preserves the same classification accuracy as the software implementation.

2.4 Implementation of the SVM learning phase

Up to now, we have not discussed the generation of the SVM learning model. In this section, we study the stability of the SVM algorithm to the selection of the learning trace and propose a community-based SVM learning architecture. An acceleration of the SVM learning process is also proposed thanks to implementation for GPU.

2.4.1 Stability of SVM to learning trace

To ensure a good accuracy of the SVM classification, the selection of training points should be performed with extreme care. In this paragraph, we study the impact of the learning trace on the accuracy of the classifier. Observing the composition per-application of real-life traffic captures, one can easily notice that it is very different from one network to another and from one time to another. Indeed, the traffic mix differs depending on whether it is a corporate network or a residential network, an access network or a core network, or different geographical zones worldwide. In our current study for instance, the three considered traces present differences in proportions of traffic classes as shown in Figure 2.3. Some applications are even absent in some traces. Furthermore, some traffic features such as inter-packet delays are clearly affected by network conditions [30] whereas others such as packet sizes should remain more stable [23] [31] [18]. Moreover, the composition of the traffic changes permanently as applications appear/disappear with time[32] or as important releases of softwares are distributed [33].

To measure the stability of the SVM algorithm, we run extensive experiments to classify traffic flows of every trace using models computed on other learning traces. In Table 2.4, the accuracy of the classification is displayed as a function of the

detection trace and the learning trace. It gives an idea of the global accuracy and the accuracy of some specific class of applications (1, 2 and 7). The results show clearly a degradation of the global accuracy when the training set is different than the detection set. For example, the accuracy of the SVM algorithm run over the Brescia trace dropped from 0.97 to 0.63 when the Ericsson trace is used for the learning phase. To deeply understand the origin of this instablity, one can notice in Table 2.4 that a difference in the proportion of Traffic classes yields an instability of the accuracy of the SVM algorithm. For example, as class 7 is not present in trace FT, using it as a learning trace for the Brescia trace, decreases the accuracy from 0.94 to 0.72.

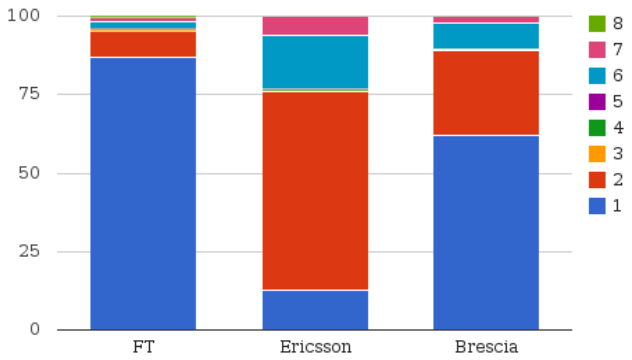

Figure 2.3: Ground truth composition of traffic traces

Traffic classes	Detection Trace	Learning Trace		
		FT	Ericsson	Brescia
All	FT	0.94	0.76	0.82
	Ericsson	0.74	0.98	0.96
	Brescia	0.71	0.63	0.97
1	FT	0.98	0.65	0.89
	Ericsson	0.88	0.97	0.86
	Brescia	0.92	0.52	1
2	FT	0.76	0.69	0.63
	Ericsson	0.63	0.99	0.87
	Brescia	0.54	0.89	0.93
7	FT	0.84	0.87	0.82
	Ericsson	0.64	0.96	0.85
	Brescia	0.72	0.91	0.94

Table 2.4: Impact of learning trace

2.4.2 A community-based SVM learning

It is then very important to adapt the composition of the learning trace to the composition of the current traffic in the network. This observation has encouraged us to implement a community-based approach to the problem of training traffic models.

To overcome the limitations of the classical classification architecture, we propose a new decentralized architecture, which principally aims to automatize the accommodation of learning traces to applications running in the network. These learning traces are obtained thanks to the collaboration of a subset of end-users. Hence, the accuracy of the classification is boosted by having up-to-date and adequate classification models. In our architecture, there are two main planes: the learning plane and the detection plane. As shown in Figure 2.4, the learning plane provides periodically a new classification model to the detection plane. The functionalities and components of these two planes are described later in this paragraph.

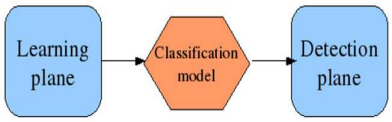

Figure 2.4: Planes of the architecture

To compute an adequate classification model, an up-to-date learning trace should be provided to the learning process. This trace should be representative of application classes which are generating the traffic in the monitored network. For this, some volunteer end-users help in constructing logs of correspondances between system calls of programs/processes and flows generated in the network with a methodology similar to the one used by the GT tool [28] of the university of Brescia. Centralizing this information in a global learning database, a trace constructor module can select the more representative and up-to-date flow records and label them with application classes. This labelling operation is done thanks to program-to-class rules updated by expert users through a learning web interface. Figure 2.5 plots the components of the learning plane and their interactions. These components are:

- System Calls Analysis Module: This module is run by some volunteer end-users and aims at logging the names and times of programs/system processes that have generated network packets.
- Learning Capture Client: this module runs on the same computer or in the same network as the System Calls Analysis module. It has the following roles. First, It captures traffic at the packet level using for example a DAG card or sniffing network interfaces. Then, it constructs flow records from packet dumps and consults the system calls logs generated by the System Calls Analysis Module in order to build a correspondence between each flow and the name of the program/ system process that has generated it. Finally, it sends to the Learning Collector Module for each flow, a message with the information

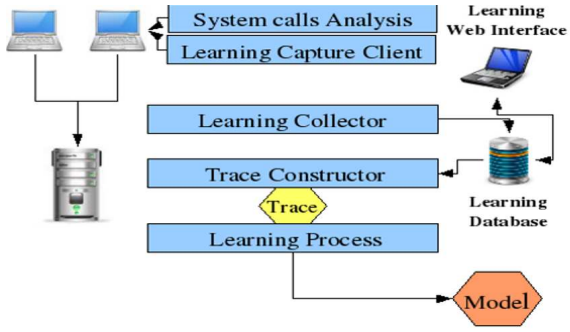

Figure 2.5: Architecture of the Learning plane

needed for the learning phase such as the size of its three first data packets, the name of the program, etc.

- Learning Collector Module: this module runs on a central dedicated server. It receives flow level records and the corresponding application names from different Learning Capture Clients and adds these records to the Learning database.

- Learning Trace Constructor: this module runs on the same machine as the Learning Process. It consults periodically the Learning database to construct an up-to-date learning trace, which it provides to the Learning process. In this operation, the constructor respects two types of rules. On one hand, it uses the program-to-class rules existing in the database to transform the program/system process names to traffic classes. These rules are updated by experts through the Learning Web Interface. On the other hand, it uses some temporal rules to select flows that better represent the current traffic classes, for example selecting the most recent flows as being the most representative of current Internet usages.

- Learning Database: this database contains all information needed for the Learning phase. It mainly includes information on traffic flows and rules to be used by the Learning Trace Constructor module. This database can then be co-located with the Learning Trace Constructor.

- Learning Web Interface: this web interface allows experts in traffic classification field to fill in program-to-class rules and up-date them. Hence, it allows to take into consideration new applications and classification strategies.

- Learning Process: this process runs on the same machine as the Learning Trace Constructor Module. It receives periodically an up-to-date trace from this module, on which it runs the SVM learning algorithm to provide a classification model. This model is written to a file that is used by the Detection Process.

2.4.3 Accelerating SVM Learning using GPU

The SVM model must be updated regularly. Indeed as new applications appear or important releases of applications are distributed the performance of the classifier can decrease dramatically. It is consequently necessary to maintain up-to-date traffic models by continuously analyzing some traffic flows generated by end-users and their associated groundtruth. One needs to run the learning process quickly to have a good adaptation of the traffic models to the traffic generated by applications.

Many solutions can be adopted to accelerate the computation of SVM traffic models. One can use many processors on the same machine or use grid computing (many machines) to balance the computing load on many available processors. Multi-core or grid computing will obviously speed-up the calibration of traffic models as they allow much faster execution than a sequential execution on a single processor. Another solution to parallelize the execution of the learning phase is to use a single machine that is equipped with a good graphical processing unit (GPU) to parallelize elementary computations.

GPU are high parallel graphic specialized processing units that can handle basic calculus. Thanks to specific languages like CUDA and OpenCL, one can write non graphic code that runs on GPUs. Although GPU are not conceived for complex operations like CPUs, GPU can run many instructions at a time and faster when they are provided with simple operations. Moreover, a cutting edge GPU is usually cheaper than a cutting edge CPU. Therefore, we decided to enhance the GPUSVM library [34] to accelerate multi-class SVM training and multi-class SVM classification using the GPU technology.

We compare the results obtained with GPUSVM to the results obtained with the LibSVM library.The testbed is a computer with a 2.66 GHz 6-core Xeon X5650 with HyperThreading enabled, 12 GB of DDR3 RAM and a Nvidia 580 GTX GPU (from the Fermi family) running a Linux 2.6.38 kernel and using a CUDA 4.0. SDK. We compared the time taken for SVM learning and SVM classification on the Brescia trace considering either an execution on the CPU with LibSVM or an execution on the GPU with GPUSVM.

In table 2.4.3, we display the obtained results in order to evaluation the acceleration factor obtained with the GPU. An acceleration of up to 48,59 times is obtained for the training phase on the Brescia dataset which is already a promising result. We think that greater acceleration factors could be obtained using multi-GPU with a FireX technology and/or improving the way operations are balanced between the GPU(s) and the (multiple) cores of the CPU. This is still to be considered as ongoing work.

2.5 Conclusion

In the monitoring architecture of VIPEER, a traffic classifier allows to understand the composition of traffic generated by end-users. This allows selection of peers with better upload/download abilities. In this chapter, we have shown that using the SVM algorithm yields good accuracy of classification when the training phase is done on a well-chosen learning trace. In this chapter, we suggested a community-

	libsvm3.1	gpusvm	acceleration
training	43 min. 47 sec.	54,08 sec.	x 48,59
classification	2 min. 57 sec.	44,52 sec.	x 3,98

Table 2.5: Acceleration with the GPU

based architecture to learn traffic models from real flows of the network. In addition to the learning architecture, we implemented a NetFPFA accelerated version of the detection phase of SVM to deal with high rate traffic when the classification is done close to end-users. Furthermore, taking into consideration the updates of the learning models, we implemented an SVM learning process that is massevily parallelized on GPU to accelerate the training phase.

As a future work, we will do further tests of the community-based learning architecture by implicating real end-users in generating traffic. This allows us to verify the stability of the SVM algorithm. Moreover, the hardware implementation of the detection algorithm will be implemented on a newer 10 Gb NetFPGA board.

A last important future task is to choose the application classes to consider for the VIPEER project. One can imagine two classes of traffic: critic traffic and non critic. Hence, one can decide whether to disturb or not a peer depending on the bandwidth consumption of the critic traffic. We suggest to modulate the available bandwidth provided to the dTracker knowing the composition of the traffic at end-users.

3 Technical specifications of the QoE Evaluater

3.1 Introduction

HTTP-based streaming is experiencing a widespread adoption by Services' Providers and CDN's operators for both mobile and fixed networks. Thus, content providers are increasingly becoming interested in evaluating the performance of such applications from the final users' perspective. Indeed, more importance is being attached to the quality as perceived by the final users, or Quality of Experience (QoE), as compared to just Quality of Service. In fact, with HTTP streaming, it is possible to have a very bad QoS, but a good QoE, since HTTP provides a reliable mean of transportation. This can happen for example when the QoS parameters are packet loss rate and packet delay. In that case, even if there will be some packet loss and delay reflecting bad QoS, the HTTP streaming, with TCP retransmissions, will still be able to provide good QoE, at least for up to some values of these parameters.

In light of the above observations, the main concerns of this section is to design a no-reference QoE monitoring module for HTTP/TCP video streaming using H.264/AVC video codec in the context of IPTV. The proposed approach uses a methodology called Pseudo-Subjective Quality Assessment (PSQA) [1, 2], which is based on Random Neural Network (RNN) [4], to estimate the QoE of H.264 streamed over HTTP. In this work, instead of packet loss pattern and latencies, we consider the Quantization Parameter (QP) used in video compression and the playout interruptions as metrics that directly impact QoE. Indeed, when using adaptive HTTP streaming the perceived quality depends directly on QP, which reflects changes in quality (i.e. change from high to low quality or vice versa), and on the playout interruptions.

Note that none of the existing approaches addresses the problem of measuring QoE with the combined case of adaptive video bitrates and using a reliable transport protocol, which is the case of the adaptive streaming over HTTP.

The remainder of this section is organized as follows. Subsection 3.2 focuses on monitoring QoE. Section 3.3 presents and discusses performance evaluation results. Finally, the section concludes in Section 3.4 with a summary recapping the main achievements of the proposed scheme.

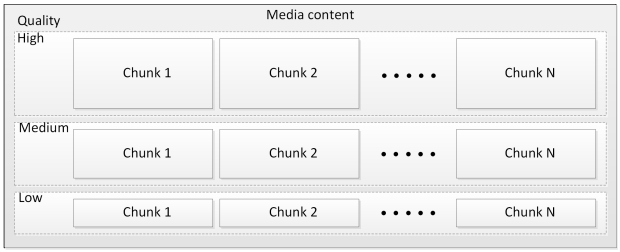

Figure 3.1: Chunk-based media content format.

3.2 QoE Estimation

3.2.1 Pseudo-Subjective Quality Assessment (PSQA)

A general technology called Pseudo-Subjective Quality Assessment (PSQA) has been proposed in [1]. PSQA is based on a specific type of queuing network used as a learning tool called Random Neural Network [4]. For every different context, such as when the video codec or the parameters affecting QoE change, a new PSQA based module is to be designed after analysing the associated parameters and after conducting new subjective tests.

The idea is to have several distorted samples evaluated subjectively by a panel of human observers. Then the results of this evaluation are used to train a RNN in order to capture the relation between the parameters causing distortion and the perceived quality. In general, the distorted sequences for a test phase are generated for a given context, and therefore a new PSQA module must be generated for every new context.

3.2.2 Adaptive HTTP Streaming

The keen interest towards multimedia streaming over the Internet, which was clearly encouraged by the development of easy-to-use content sharing platforms (e.g. YouTube), is making HTTP/TCP streaming the leading technology in the media delivery sectors [5] for both mobile and fixed networks. As opposed to the former protocols, HTTP enables a reliable and an adaptive streaming process. These properties are directly inherited from those of the TCP protocol. It also allows to seamlessly bypass firewalls and adapt the streaming quality to the bandwidth, which makes the technology particularly interesting for a wide deployment. For adaptive bitrate streaming, the media file to be streamed is fragmented into small segments or chunks (see Figure 3.1) of same duration (e.g. a few seconds) [6].

In order to allow adaptive streaming, each chunk is decoded independently. This enables seamless switching from one quality to another when network conditions change. This is because once the playout of a chunk is finished, the video player can start playing the next chunk of different quality as each chunk is independently decodable.

Table 3.1: Notations

Symbol	Definition (values are implied to be measured over a fixed interval)
D_{avg}	Average value of pauses or interruptions
D_{max}	Maximum pause duration
N	Number of playout interruptions
QP	Quantisation paramter
α	1 - D_{avg}/D_{max}
MOS	Mean Opinion Score

3.2.3 Input parameters for QoE evaluator

In order to use PSQA for Adaptive HTTP Streaming, first the relevant parameters need to be identified and their effect on the perceived quality needs to be studied. In the next step, the important parameters will have to be simulated, which will result in distorted video sequences. These distorted video sequences will be used to train the PSQA tool with the help of a panel of human observers. The trained PSQA will, then, be used in real time to estimate the subjective video quality. In the following text, we describe the parameters that are considered for QoE estimation in the context of HTTP streaming over TCP/IP networks (see [7] for more details). The definitions of the parameters are also provided in Table 3.1.

It should be noted that other parameters, not described below, like resolution and frame rate are either constant in our study or, like losses, delay and jitter that cause packets to miss their deadline, are converted into playout interruptions.

3.2.3.1 Playout Interruptions

TCP/IP networks are characterised by frequent packet losses and fluctuating bandwidth over time. In opposition to the streaming over UDP, the streaming over HTTP implies an automatic recovery of lost packets, handled by the TCP protocol. This clearly eliminates video distortions in most of the players. However, the packets retransmissions introduce more overhead and latencies, which, as well as bandwidth fluctuations, may cause playout interruptions. Thus, playout interruptions should be taken into account for QoE estimation.

We model the playout interruptions using three parameters observed during a measurement interval containing a fixed duration of original video data (16 seconds in this paper, but total time can be longer due to the presence of interruptions): the total number of playout interruptions N, the average of interruption delays (that means video pauses) D_{avg} and the maximal interruption delays D_{max}. These parameters are normalized with the video sequences lengths to make the QoE module more robust to configuration changes. Note that, for longer videos, a different QoE score will be provided every 16 seconds which in turn is the granularity of the estimation.

3.2.3.2 Quantization Parameter (QP)

In the context of HTTP streaming, another parameter that significantly impacts the QoE is the amount of video compression that in turn is controlled by the quanti-

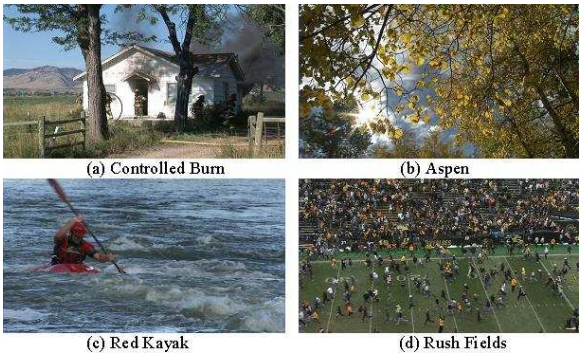

Figure 3.2: Videos used for subjective testing. Source: ITU VQEG

zation. H.264 codec uses a quantization parameter (QP) to quantize the transform coefficients obtained while encoding the video. QP ranges from 0 to 51 and the trade-off is that a high value of QP means more loss of information and lower quality, but it also means lower bitrate; and vice versa.

For QoE estimation, we consider the average of QP values, QP_{avg}, over all MBs in all video frames present over the time window considered for evaluation.

3.3 QoE Evaluator

In order to generate the QoE Estimation module, 4 different video sequences of 16 seconds each were considered, as shown in Figure 3.2. The resolution is 720p, fps = 30, GOP size = 60 frames. The high profile of H.264 is used. The encoder used is x264 [9]. The value of QP was varied from 22 to 48.

A video database was generated by simulating different combinations of the input parameters described in previous section. The videos were then evaluated by a panel of 15 users using single stimulus (SS) testing methodology [8]. A MOS scale of 1, very bad, to 5, excellent, was used. The MOS scores and the corresponding values of the input parameters were then used to train RNN. The trained RNN is used as the QoE Evaluator as shown in Figure 3.3. More details about the RNN conception is given in [7].

The results are shown in Figures 3.4, and 3.5, which show the estimated QoE with respect to different pairs of parameters, the remaining ones being fixed. Figure 3.4 shows that users are more sensitive to video playout interruptions as compared to an increase in QP when the value of QP is low. When QP increases or the quality degrades due to increased quantization, initially the QoE scores fall very slowly. Only after reaching a high value of QP, the QoE scores start to decrease faster. Whereas, QoE drops faster with increasing D_{max}, initially, but after a higher value of D_{max} the decrement of QoE becomes slow. This is because after a high value of D_{max}, around 6 to 8 seconds, the dropped QoE becomes saturated and the users are less sensitive to further increments in D_{max}.

Besides, increasing the value of QP decreases the video bitrate. Thus, when the bandwidth decreases in the network, the QP can be increased a lot, to adapt the streaming data rate to the available bandwidth, rather than risking even a single

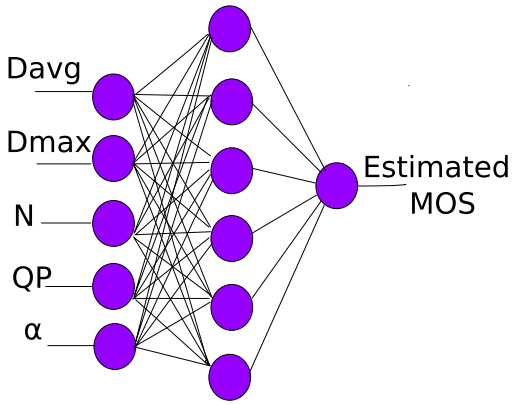

Figure 3.3: QoE Evaluator based on RNN.

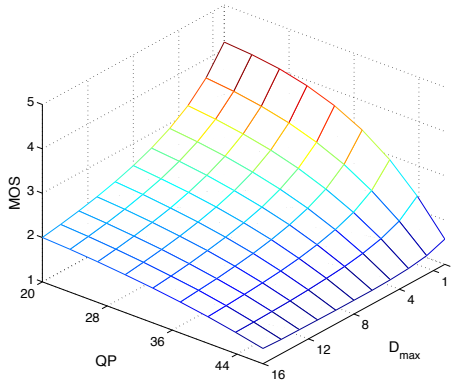

Figure 3.4: MOS vs QP and D_{max}. With $1 - \alpha = 0.9$ and $N = 2$.

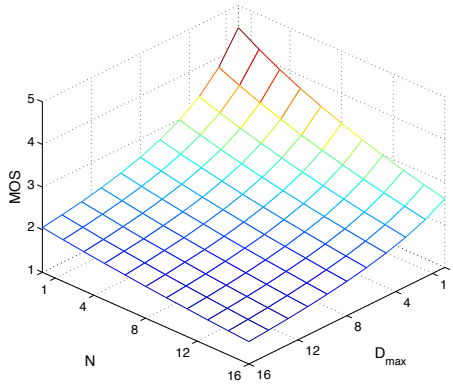

Figure 3.5: MOS vs D_{max} and N. With $QP = 23$, $1 - \alpha = 0.9$.

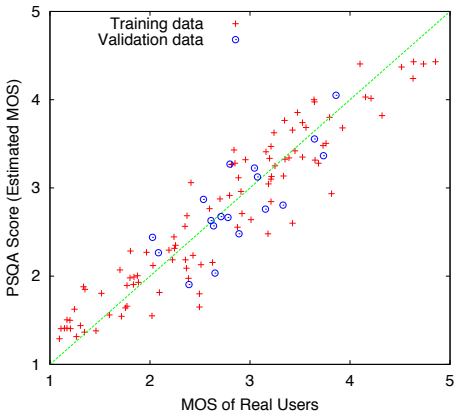

Figure 3.6: Real vs Estimated QoE scores with our QoE module.

playout interruption.

With respect to above observations, our QoE model can be integrated with the controller of adaptive HTTP streaming. It will help the controller to make decisions that optimize QoE by proving an estimate of QoE for a given value of QP and network parameters.

Figure 3.5 shows the QoE with respect to D_{max} and N. It can be seen that for lower values of D_{max} or N the QoE is very sensitive to both. However, for higher values, QoE decreases very slowly because after a certain value of D_{max} or N the quality is already bad enough and users are not sensitive to further increments. Also note that the worst value of predicted MOS in this figure is 1.5, but for such high values of D_{max} and N the MOS should be 1.0, that is the minimum MOS possible. This prediction error of 0.5 is because RNN shows saturation near bad quality and because while training we wanted to be accurate when quality is good instead of being accurate when quality is already bad.

Figure 3.6 shows the scatter plot with estimated MOS versus real MOS obtained from the subjective tests. The points corresponding to the training as well as the validation data are shown. The scatter plot shows the good accuracy of the estimation. This is also reflected by the overall Root Mean Square Error (RMSE) of 0.36 for all data (with slightly higher and lower RMSE for training and validation sets, respectively) on the MOS scale going from 1.0 to 5.0. The RMSE is less than that of the human test panel where it was 0.59 and thus it is satisfactory.

3.4 Conclusion

In this section, we have addressed the problem of estimating the QoE of video streaming in TCP/IP networks. As a solution, we designed an automatic no-reference QoE estimation module for HTTP video streaming using TCP and H.264 video codec and the trained RNN function is provided in this paper. The proposed approach is different from the existing ones as it addresses the problem of measuring QoE in the combined case of adaptive video bitrates and the use of a reliable transport protocol. This is the case of the adaptive streaming over HTTP. Exten-

sive simulations showed that our model accurately measures the QoE and performs better than an existing QoE model when the value of QP is varied.

4 Tools' Integration within the testbed

4.1 Architecture reminder

Our goal is to integrate several functions in the measurement platform:

1. real-time determination of the network parameters such as the available bandwidth, the packet loss rate, the jitter and the delay by actively probing the access link,

2. determination of the activity of the end-user in real-time that is to say the composition of traffic generated by this end-user in terms of applications,

3. automatic estimation of the Quality of Experience that can be delivered to the different end-user applications, taking into account the network state in real time.

In deliverable D2.1, we have described the corresponding architecture for monitoring. The picture 4.1 is a summary of it.

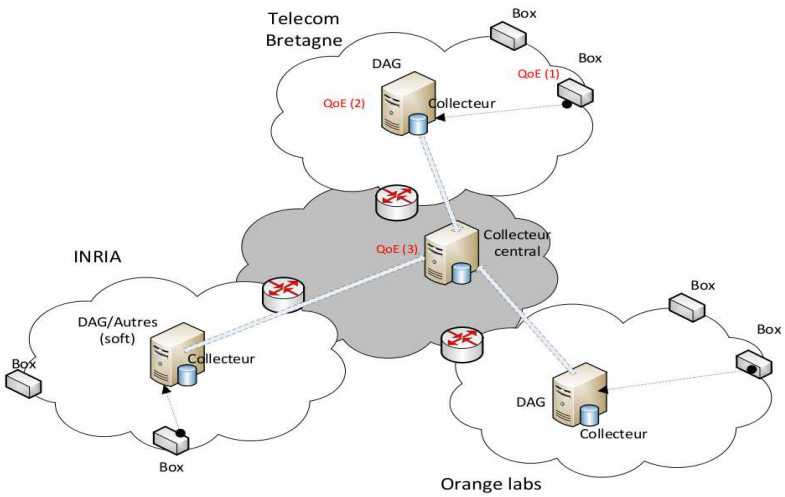

Figure 4.1: Proposed architecture

As mentioned in this picture, locally we have some probes with DAG cards that collect the traffic (passive measurements). Then, they send theirs results to the dCollector (a central collector that will run the classification of Nicofix for example). For more detail, see D2.1.
Now we will focus on communication between the different entities that composed the monitoring and in the last section of this chapter, we will describe the link with the global architecture of VIPEER.

4.2 Communication between the different entities

Probes with DAG cards measure some traffic characteristics. Then they send this information to the dCollector. Here we explain in detail this step.
First, on the dCollector, a socket is open to communicate with the probes. We have chosen the UDP protocol and the number 9999 for the associated port. This number could be everything else but we have to define one so that we could block others UDP traffic not expected (for security reason). Then others entities could send periodically (say every 60 seconds) or when needed reports on this open channel. This is for the channel of communication, and now we detail the content of these exchanges.

For the Nicofix part, the dCollector need some information to run the classification of flows:
- the source IP address,
- the destination IP address,
- the source port,
- the destination port,
- the protocol number,
- the size of first, second and third packet.

For evaluation and demonstration issues, a step of programming is needed. The dCollector is then written in C language. So we have the corresponding structure for Nicofix classification:

```
struct classif_flow {
      uint32_t dst_ip;
      uint32_t src_ip;
      uint16_t src_port;
      uint16_t dst_port;
      uint8_t proto;
      uint32_t nb_bytes1;
      uint32_t nb_bytes2;
      uint32_t nb_bytes3;
};
```

For QoE purpose, the dCollector store those parameters:
- the IP address of the client (the evaluation of QoE is for him)
- the MOS score,

- the QP parameter,
- the playout interruptions number,
- the playout interruptions time in s.

Here is the corresponding structure in C language:

```
struct qoe_flow {
        uint32_t qoe_nb;
        uint16_t ad_client;
        uint8_t mos ;
        double qp;
        uint32_t nb_playout;
        double time_playout;
        double date;
};
```

Finally, the dCollector store the monitoring metrics like available bandwidth, delay...

We see that there are different types of information coming at the dCollector. To know which kind arrives we have to introduce a structure in packet exchanges. Here is the corresponding structure in C language:

```
//vipeer export packet format:
typedef struct vipeer_packet {
        uint32_t probe_id; // to know from which probe the data come from
        uint8_t packetType; // to know the type of information: 1,2 or 3
        uint16_t data_length; //if needed...
        union {
                struct classif_flow cf; //type 1: Nicofix classification
                struct qoe_flow qf; //type 2: QoE
                struct metrics m; //type 3: Monitoring metrics
        } data;
} V_PCK;
```

With this structure, the dCollector knows the king of information (classification, QoE or monitoring metrics) and then, it knows how to interpret the data. We have an extra information here: we know from which probes the information arrives and this could be useful to aggregate some data by localisation (for example, if a lot of clients have bad QoE in a same area an alarm could be done for this region).

After receiving the data, the dCollector could compute them to give results (like for the Nicofix classification) and finally, the dCollector has to store all needed results. Now, we will see the corresponding database for this purpose. As the classification, the QoE evaluation and the monitoring metrics are independents, a dedicated table is built for each of these goals. We begin with the table named TAB_classif dedicated to the classification of user applications, in figure 4.2.

Champ	Type	Interclassement	Attributs	Null
flow_id	int(10)		UNSIGNED	Non
ad_source	varchar(16)	latin1_swedish_ci		Non
ad_destination	varchar(16)	latin1_swedish_ci		Non
port_source	varchar(8)	latin1_swedish_ci		Non
port_destination	varchar(8)	latin1_swedish_ci		Non
protocol	tinyint(3)		UNSIGNED	Non
nb_bytes1	int(10)		UNSIGNED	Non
nb_bytes2	int(10)		UNSIGNED	Non
nb_bytes3	int(10)		UNSIGNED	Non
type	tinyint(3)		UNSIGNED	Non
date	double			Non
ip_probe	varchar(16)	latin1_swedish_ci		Non

Figure 4.2: TAB_classif

We have the needed data for the classification and we have the result of it in the field type with a number. In figure 4.3, we have the correspondence between the number and an understandable text.

result	nom
1	Web & News
2	P2P download
3	Direct download
4	Streaming
5	Game
6	Mail
7	Instant messaging & VoIP
8	Distant control(SSH)

Figure 4.3: Type_classif

For the QoE, in table 4.4 named TAB_qoe, we have all the needed information.

Champ	Type	Interclassement	Attributs	Null
qoe_nb	int(10)		UNSIGNED	Non
ad_client	varchar(16)	latin1_swedish_ci		Non
mos	tinyint(3)		UNSIGNED	Non
qp	double			Non
nb_playout	int(10)		UNSIGNED	Non
time_playout	double			Non
time	int(10)		UNSIGNED	Non
date	double			Non

Figure 4.4: TAB_qoe

All the tables are needed for ihm purpose too. For a supervisor, it is better to have a view on graphics for example than an access to a database. So, in the dCollector, we have a web site with these results. For demonstration purpose this web site is open on the internet but with a secure access thanks to password. In 4.5 and 4.6 we have some examples of this friendly interface.

With the web site on the dCollector, we could have a look at results from WP2, but these results are needed by the rest of the VIPEER project so we will explain now how they take part in the global architecture.

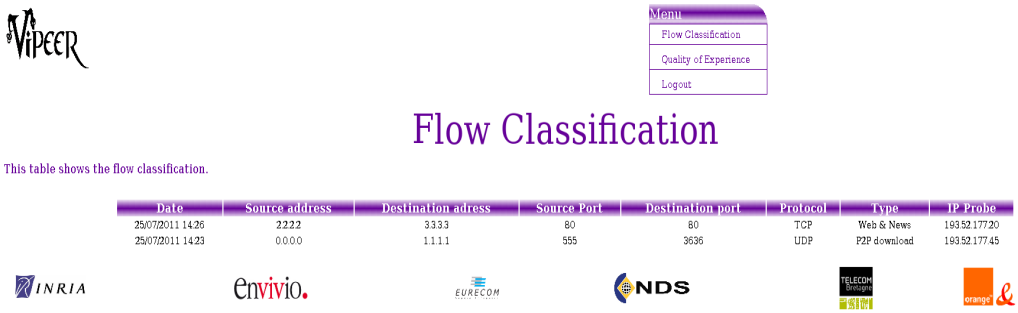

Figure 4.5: Web page for classification

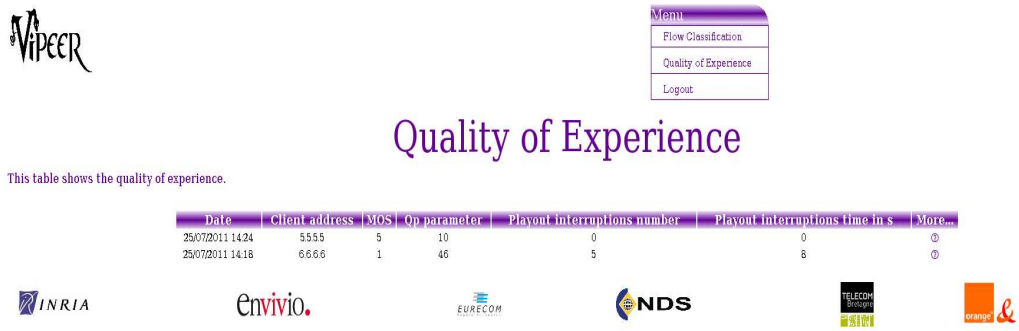

Figure 4.6: Web page for QoE

4.3 Global architecture

In deliverable D1.2, there is a view of the global architecture when a user asks for content, from the manifest negotiation to the search of content. The figure 4.7 resume this step.

As we see, the dTracker is the entity that drives the control plane by taking the decision in the content domain: which content, where, how many representation of it...

As mentioned before, all results from WP2 are stored in the dCollector and all these results are in the control plane. So it is natural that dCollector and dTracker will communicate. Two kinds of communication are considered:

1. dTracker ask the information at the dCollector when it has to take decision. As the dCollector has database with the expected results, one can imagine that the dTracker will ask by SQL request the online database on the dCollector.

2. dCollector compute some functions like the Nicofix classification or QoE synthesis. If a part of the network has very bad conditions (poor QoE or very low bit rate,...), the dCollector will know the situation and could warm the dTracker. This second type could be named on events.

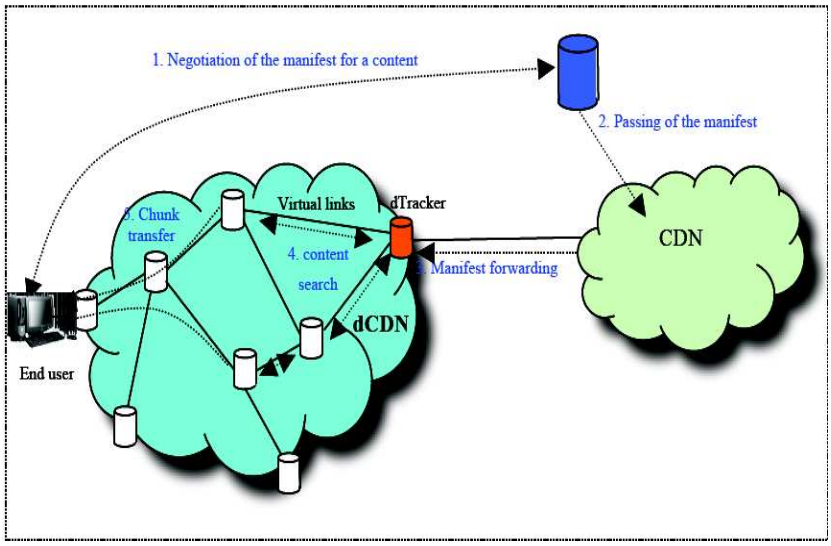

Figure 4.7: Manifest negotiation and content search

4.4 Conclusion

The figure 4.8 is a good summary of the communication between dTracker and dCollector.

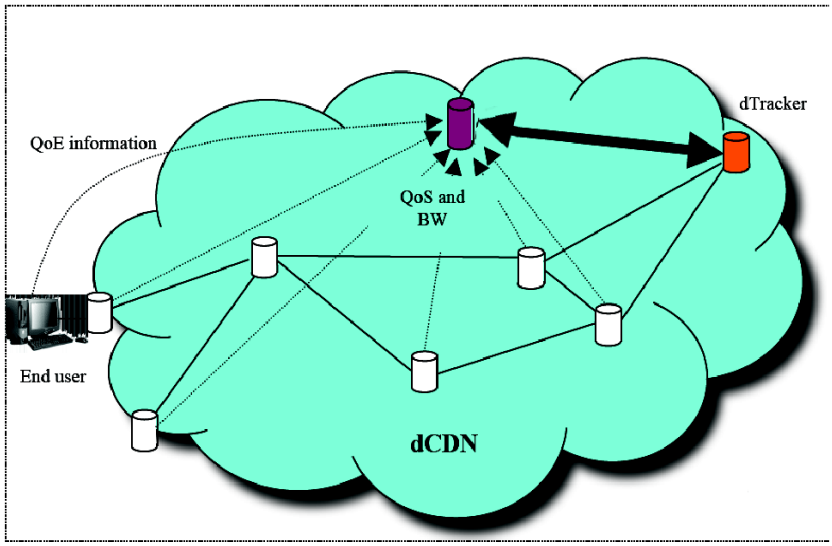

Figure 4.8: Global architecture with dCollector

We can note that it is easy to implement the link between the WP2 and the rest of the VIPEER project because all is concentrate on communication between dCollector and dTraker.

5 Conclusions

We presented in this deliverable the technical specifications of the tools and modules proposed within the WP2. These tools will clearly help to improve the performance of the considered ViPeer framework by not only serving as a global indicator of network performance and customer satisfaction, but also by making long term decisions, such as the network planning. Meanwhile, this monitoring could also help in determining both the optimal storing strategy and the best reaction to services' quality degradation. Besides, this will help to optimize the distribution strategy of the dCDN so as to minimize the impact of the dCDN on the QoS perceived by the set-top-box owners and to maximize the QoS experienced by the dCDN's users.

Bibliography

[1] S. Mohamed and G. Rubino, "A Study of Real–time Packet Video Quality Using Random Neural Networks," *IEEE Trans. On Circuits and Systems for Video Tech.*, vol. 12, no. 12, pp. 1071–1083, Dec. 2002.

[2] G. Rubino, "Quantifying the Quality of Audio and Video Transmissions over the Internet: The PSQA Approach," in *Design and Operations of Communication Networks: A Review of Wired and Wireless Modelling and Management Challenges*, edited by J. Barria, Imperial College Press, 2005.

[3] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," Neural Comput., vol. 1, pp. 502-511, 1989.

[4] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," Neural Comput., vol. 1, pp. 502-511, 1989.

[5] N. Zong, "Survey and Gap Analysis for HTTP Streaming Standards and Implementations", Internet-Draft: draft-zong-httpstreaming-gap-analysis-01, October 2010.

[6] Q.Wu, "Problem Statement for HTTP Streaming", Internet-Draft: draft-wu-http-streaming-optimization-ps-01, Sept. 2010.

[7] K. Singh, Y. Hadjadj-Aoul and G. Rubino, "Quality of Experience estimation for Adaptive HTTP/TCP video streaming using H.264/AVC," *In proc of IEEE CCNC 2012*, Las Vegas, Nevada, USA, Jan. 2012.

[8] ITU-R Recommendation BT.500-11, "Methodology for the subjective assessment of the quality of television pictures", June 2002.

[9] x264, "http://www.videolan.org/developers/x264.html".

[10] Internet2, : One-Way Ping (OWAMP) http://www.internet2.edu/performance/owamp/, january 2009

[11] D. Mills, U. Delaware, J. Martin, W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification". IETF RFC 5905, June 2010.

[12] IEEE 1588-2008, "Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". 2008

[13] Corinna Cortes and Vladimir Vapnik,Support-Vector Networks, Machine Learning, p. 273-297, 1995.

[14] M. Canini and W. Li and M. Zadnik and A.W. Moore, Experience with high-speed automated application-identification for network-management, Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '09,2009

[15] A. Este and F. Gringoli, On-line SVM traffic classification, Proceedings of the 7th International Wireless Communications and Mobile Computing Conference, IWCMC, 2011, Istanbul, Turkey.

[16] A. Este and F. Gringoli and L. Salgarelli , Support Vector Machines for TCP traffic classification", Computer Networks , , . 2476 - 2490 , 2009

[17] DAHMOUNI Hamza,VATON Sandrine,ROSSÉ David,ineNet 2007 : ACM Sigmetrics Workshop on Mining Network Data, 12 june 2007, San Diego, United States, 2007, pp. 29-34

[18] A. Este, F. Gringoli and L. Salgarelli,, On the stability of the information carried by traffic flow features at the packet level, ACM SIGCOMM Comput. Commun. Rev, New York,2009.

[19] G.Gomez and P.Belzarena, Early Traffic Classification using Support Vector Machines, Fifth International Latin American Networking Conference, LANC'09, Pelotas, Brazil.

[20] P. Bermolen and M. Mellia and M. Meo and D. Rossi and S. Valenti, Abacus: Accurate behavioral classification of P2P traffic, Elsevier Computer Networks, volume 55, pp. 1394-1411, 2011

[21] D. Rossi and S. Valenti, Fine-grained traffic classification with netflow data, Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, IWCMC '10, Caen, France.

[22] N. Williams and S. Zander and G. Armitage, A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification, ACM SIGCOMM Computer Communication Review, 2006

[23] Hyunchul Kim and Dhiman Barman and Michalis Faloutsos and Marina Fomenkov and Kiyoung Lee, Internet Traffic Classification Demystified: The Myths, Caveats and Best Practices, In Proc. ACM CoNEXT, 2008

[24] P. Bermolen and M. Mellia and M. Meo and D. Rossi and S. Valenti, Abacus: Accurate behavioral classification of P2P traffic, Elsevier Computer Networks, volume 55, pp. 1394-1411, 2011.

[25] LIBSVM: a library for Support Vector Machines, `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[26] Corinna Cortes and Vladimir Vapnik, Support-Vector Networks, Machine Learning, pp. 273–297, 1995.

[27] l7-filter: application layer packet classifier for Linux, http://l7-filter.clearfoundation.com/

[28] F. Gringoli and L. Salgarelli and M. Dusi and N. Cascarano and F. Risso and K.C. Claffy, GT: picking up the truth from the ground for Internet traffic, ACM SIGCOMM Computer Communication Review, volume 39, pp. 13-18, 2009.

[29] NetFPGA: a line-rate, flexible, and open platform for research, and classroom experimentation, http://netfpga.org/

[30] M. Jaber and R. Cascella and C. Barakat, Can we trust the inter-packet time for traffic classification?, Proceedings of the IEEE International Conference on Communications (ICC), 2011, Kyoto, Japan

[31] T. Nguyen and G. Armitage, A survey of techniques for Internet traffic classification, IEEE Communications Surveys and Tutorials, volume 10, pp.56-76, 2008.

[32] W. Li and M. Canini and A.W. Moore and R. Bolla, Efficient Application Identification and the Temporal and Spatial Stability of Classification Schema, Computer Networks, Special Issue on Traffic classification and its applications to modern networks, vol 53, pp. 790–809, 2009.

[33] D. Rossi and S. Valenti, Fine-grained traffic classification with netflow data, Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, IWCMC '10, Caen, France.

[34] Catanzaro, Bryan Christopher and Sundaram, Narayanan and Keutzer, Kurt , Fast Support Vector Machine Training and Classification on Graphics Processors, EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2008-11, 2008