Classification of HTTP traffic based on C5.0 Machine Learning Algorithm

Tomasz Bujlow, Tahir Riaz, Jens Myrup Pedersen
Section for Networking and Security, Department of Electronic Systems
Aalborg University, DK-9220 Aalborg East, Denmark
{tbu, tahir, jens}@es.aau.dk

***Abstract*—Our previous work demonstrated the possibility of distinguishing several groups of traffic with accuracy of over 99 %. Today, most of the traffic is generated by web browsers, which provide different kinds of services based on the HTTP protocol: web browsing, file downloads, audio and voice streaming through third-party plugins, etc. This paper suggests and evaluates two approaches to distinguish various types of HTTP traffic based on the content: distributed among volunteers' machines and centralized running in the core of the network. We also assess the accuracy of the centralized classifier for both the HTTP traffic and mixed HTTP/non-HTTP traffic. In the latter case, we achieved the accuracy of 94 %. Finally, we provide graphical characteristics of different kinds of HTTP traffic.**

***Index Terms*—traffic classification, computer networks, HTTP traffic, browser traffic, C5.0, Machine Learning Algorithms (MLAs), performance monitoring**

I. Introduction

The assessment of the Quality of Service (QoS) in computer networks is a challenging task because different kinds of traffic flows (voice and video streaming, file download, web browsing) have different requirements. Therefore, to estimate the performance, we need to know what type of data flow is currently being assessed. There are many methods for distinguishing computer network traffic, including the classification by ports, Deep Packet Inspection (DPI), or statistical classification [1]. We compared them in [2] and we assessed that these methods are not sufficient for the real-time identification of HTTP traffic.

We had two possible approaches to classify the flows in a high-speed computer network infrastructure: centralized and distributed. We implemented the distributed approach as the Volunteer-Based System (VBS) and presented in [2]. VBS clients installed on users' computers collect the data together with the name of the corresponding application. The necessary statistical parameters are calculated on the client side and sent to the database server. We designed the centralized solution as a flow-examining-application installed in any point of the network. All the flows passing through that point are captured and assigned to a particular application class by the C5.0 Machine Learning Algorithm (MLA) [3]. As training data, we used the data collected by VBS. The proposed design of a solution for estimating QoS using both these approaches and combining passive and active measurements was described in [4]. The accuracy of the distributed QoS assessment solution is approaching 100 %, as it uses the process names taken directly from the system sockets during the classification. The accuracy of our centralized QoS assessment was assessed to be 99.3–99.9 %, due to the C5.0 classification error estimated based on our previous approach to classify 7 different applications [3].

In previous papers, we assumed that one application carries only one type of traffic and, for this reason, we took into account only applications fulfilling this criterion. However, the data collected by VBS showed that nowadays majority of traffic is generated by HTTP-based applications as web browsers. Until now, we were treating this kind of traffic as a general *web* traffic class, which in effect consisted of interactive traffic (web pages), audio and video streams, and big file downloads (including big video files downloaded directly from a website by the user or downloaded indirectly by a web player, as YouTube). Flows carrying different kinds of content can have different characteristics and QoS requirements [5] and, therefore, they need to be distinguished and processed in different ways. The measured characteristics of different content types found within HTTP flows are shown at the end of this paper. All these factors lead to the conclusion that during QoS assessment, we are interested in the type of the traffic (taking into account both the type of the content and the type of the content delivery, as streaming or casual download), not in the application which it generates. In this paper, we present and evaluate a method for recognizing different kinds of HTTP traffic.

Other methods for the classification of HTTP traffic are shown in [6] and [7]. In [6], the authors propose to use the size of the flow and the number of flows associated with the same IP address to determine the character of the traffic by 3 different MLAs. Unfortunately, this approach requires to have the traffic collected in advance, and in consequence, it is not suitable for the real-time classification needed for QoS assessment purposes. The method described in [7] is based on keyword matching, flow statistics and a self-developed algorithm. This approach also does not fulfill our needs because it requires processing entire flows: first to match the signature, then to extract statistics, such as the number of packets contained by the flow. As opposite, our centralized solution is able to classify the data based on 35 consecutive packets from a random point of a flow. As a consequence, we can monitor flows very quickly.

The remainder of this paper gives the overview of our solutions for the distributed and centralized classification of

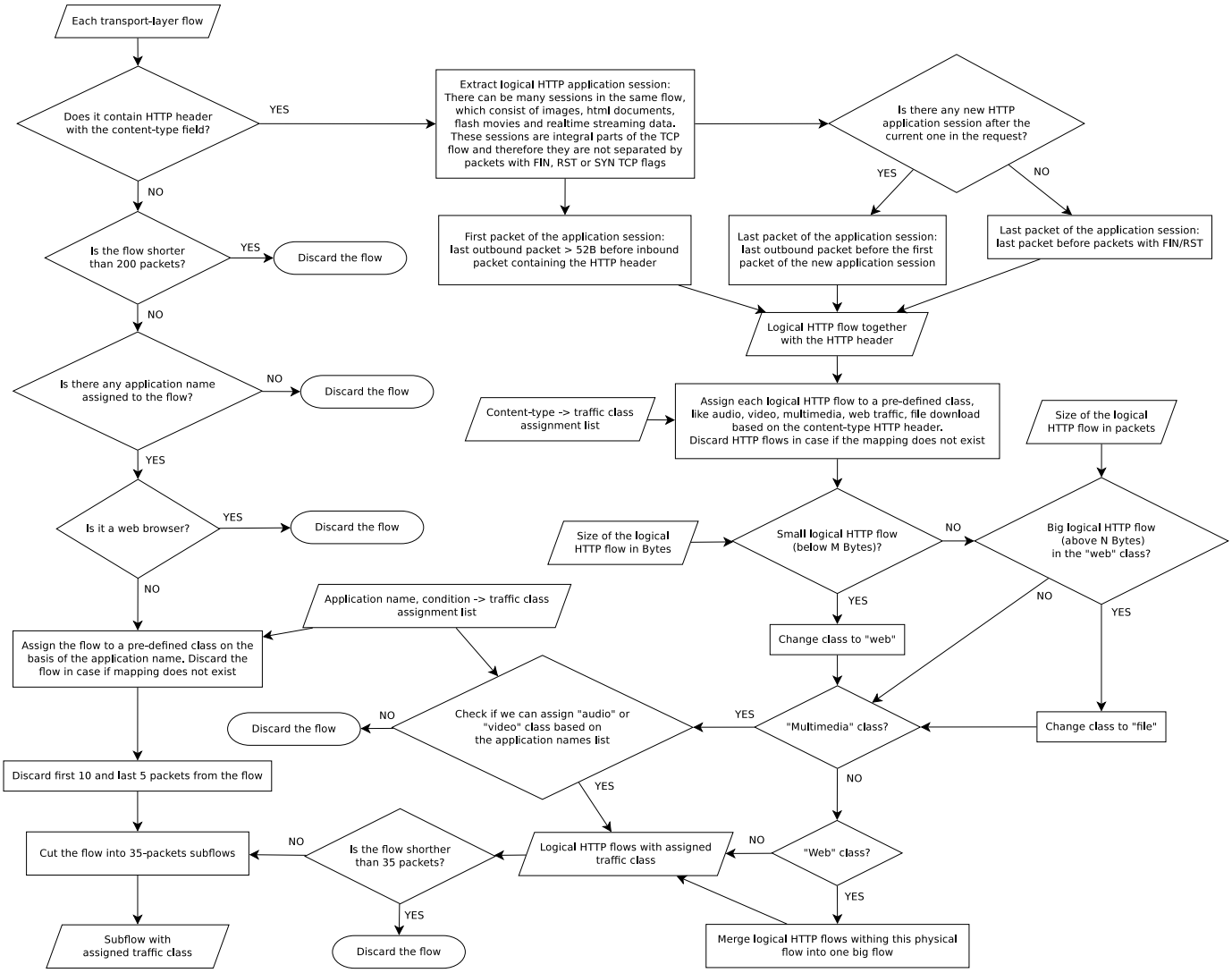

Figure 1. Overview of the method for obtaining the training data

network traffic and our methods for providing precise input data, describes the results, and shows various traffic profiles. We assessed the accuracy of the classification while changing parameters in the algorithm. The data used in our experiments originate from 5 private machines running in Denmark and in Poland as well as 18 machines installed in computer classrooms in *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*, a high school in Poland.

II. Centralized classification method

We designed the centralized method to be used in the core of the network. 35-packet long snippets from the selected flows are inspected by the statistics generator, which calculates the values of the relevant parameters. Based on the calculated statistics, the C5.0 MLA is able to predict the traffic class of the flow. The first and the most important issue in our solution was how to train the classifier properly. As a consequence, we designed and implemented an algorithm, which uses pre-classified browser traffic to generate training cases for different classes of traffic. The description of the algorithm is shown in Figure 1.

Browser traffic can be classified based on two different approaches: by using HTTP headers, or application names and additional flow conditions like ports. Table I contains examples of different services accessible by Firefox web browser, together with the information about the chosen method of classification. As shown, the type of the content delivered by most services can be classified accurately by the *content-type* field in the HTTP header. Unfortunately, in some cases, we are not able to distinguish HTTP audio from HTTP video streams (as shown in the case of *application/x-mms-framed* content type, used both for streamed audio and video content). However, streamed multimedia content is often played by plugins which use the Real-Time Messaging Protocol (RTMP) instead of HTTP, so the content can be

Table I
EXAMPLES OF DIFFERENT SERVICES PROVIDED BY WEB BROWSERS

Media	Location	App. name	Content type	Classification
Internet radio				
The Voice	http://www.thevoice.dk/popup/popup.php?tab=radio	firefox	audio/mpeg	By content type
NOVAfm	http://www.novafm.dk/popup/popup.php?tab=radio	firefox	audio/mpeg	By content type
Radio 3	http://www.radio3.dk/sites/all/modules/netplayer/player.php	plugin-container		Impossible
RMF FM	http://www.rmfon.pl/play,5	plugin-container	audio/aacp	By content type
ESKA	http://www.eska.pl/player?streamId=101	firefox	audio/mpeg	By content type
CNN radio	http://radioradio7.com/radio/CNN.html	totem-plugin-	application/x-mms-framed	By content type
Embedded audio				
Wrzuta.pl	http://www.wrzuta.pl	firefox	audio/mpeg	By content type
Video on Demand				
Youtube	http://www.youtube.com/	firefox	video/x-flv	By content type
Ipla	http://www.ipla.pl	iplalite	video/x-flv	By content type
Onet Video News	http://www.onet.pl	firefox	video/x-flv	By content type
CNN Video News	http://edition.cnn.com/video/	firefox	video/x-flv	By content type
Wrzuta.pl	http://www.wrzuta.pl	firefox	video/mp4	By content type
Internet TV				
Justin.tv	http://www.justin.tv/	plugin-container		By app name and remote port 1935
Al-Jazeera	http://www.aljazeera.com/watch_now/	plugin-container		By app name and remote port 1935
PDR	http://www.pdr.pl	totem-plugin-	application/x-mms-framed	By content type
File download				
File 1	http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-solaris-sparc.tar.Z	firefox	application/x-compress	By content type
File 2	http://www.skatnet.dk/test/testfile.avi	firefox	video/x-msvideo	By content type as *video*. However, the most proper class would be *file download*

separated using plugin names (such as *plugin-container*) and RTMP remote port (1935). From the QoS point of view, the problem is that based on the *content-type* field, we cannot distinguish streamed multimedia content from multimedia files embedded on websites (such as YouTube) and just downloaded in the background to the user's computer, because they can use identical values of the *content-type* field, for example, *audio/mpeg*. The same situation happens when a user downloads a video or audio file explicitly by using a download link. Therefore, for the purpose of this experiment, we placed all the flows delivering the *video* content to the same class, regardless, if the content was streamed or downloaded.

As the first step, we need to decide if we are dealing with an HTTP-based flow or another kind of transport-layer flow. For this purpose, we examine each packet in the flow and check if the HTTP header exists. If yes, we look for the *content-type* field. If we can obtain the information, the preferred way of processing is always to handle the flow as an HTTP-based flow, as it allows to recognize different kinds of flows generated by one application. Short flows (below 200 packets in the case of regular flows, and below 35 packets in the case of HTTP-based flows) are discarded because they seem to be less useful from the QoS measurement point of view.

A. Regular transport-layer flows

Regular flows are processed based on the assignments between the application names and traffic classes. Most applications are specialized to handle specific types of traffic (voice conversations for Skype, file transfer for FTP clients, or interactive traffic for games), but they also generate background traffic. For example, Skype shares a distributed users' directory, free file transfer clients tend to download advertisements to display them on the screen when doing their job, and games have control connections to the main server. These flows, acting as noise, are usually quite short. To eliminate their impact, we decided to discard all flows shorter than 200 packets. If there is no application name assigned to the flow, the flow is discarded. Flows associated with HTTP-based applications (like web browsers) are discarded as well, because they are not recognized as HTTP flows and their type is unknown. Then the flows are checked against the assignments between the application names and traffic classes. If we cannot find any match, the flow is discarded as well. As it is written in [2], around the first 10 and the last 5 packets of each flow have different characteristics of size parameters than the other packets. As a result, these packets are cut out of the flow. Next, the flow is split into 35-packet subflows, which are provided to the statistics generator. The generated statistics are given as the input to the C5.0 classifier as training or test data. It was shown in [2] that further increasing of the number of packets in the subflow does not improve significantly the accuracy of the classifier. Using the reasonably smallest number of flows allows to perform faster

traffic classification and saves system resources, what allows to process more flows at a time.

B. HTTP-based transport-layer flows

Dealing with HTTP-based flows is more complex, as one transport-layer flow by HTTP can carry multiple different files as text, images, audio, and video. For this reason, we split the transport-layer flow into entities carrying different files or streamed content (called later *separate HTTP flows, as they have separate HTTP headers*), which are mapped to a traffic class based on the *content-type* field in the HTTP header. We found that the *content-type* field in the HTTP header is present in and only in the first inbound packet of a new logical HTTP flow. If the mapping does not exist, the HTTP flow is discarded.

We decided to specify the following traffic classes: *audio*, *file download*, *multimedia*, *video*, and *web*. The *multimedia* class was assigned to traffic with content-types, which could carry audio as well as video (regardless if the content was streamed or downloaded, as based on the content it is impossible to detect). The *file download* class was assigned to big file downloads (however, except the multimedia files, which were assigned to one of the multimedia classes), and the *web* class to the traffic produced by interactive web browsing. It was very hard to define what the interactive web browsing is, but we decided to create this class as characteristics of transport-layer flows carrying multiple small files like HTML documents, web images and stylesheets are different than the characteristics of downloads of big files. It can be even more complicated because small few-second videos and animations on websites behave more like the interactive traffic than the real video traffic. On the other hand, big images embedded on websites behave like file downloads. Therefore, we decided to consider all the small (below M Bytes) HTTP flows as web interactive traffic, and all the big *web* flows (above N Bytes) as *file download* traffic. Values for M and N are intended to be chosen experimentally. All the interactive HTTP *web* flows within the transport-layer flow were merged together into one big *web* flow. It makes no sense to calculate statistics for each small web element (HTML file, images, etc) separately, because they are visible across the network as one interactive flow, so they must be processed holistically to assure the proper assessment of the QoS level.

The entities carrying the *multimedia* content must be re-classified as either *audio* or *video*, because they have different characteristics and requirements. We use the assignments between the application names and traffic classes to see if the application assigned to that flow is purely audio or video oriented. If not, the flow is discarded. Flows shorter than 35 packets are dropped to ensure compatibility with processing regular transport-layer flows. Finally, the flow is split into 35-packets subflows, which are provided to the statistics generator. The generated statistics are the input given to the C5.0 classifier as the training or the test data.

III. Data sources

The algorithm of obtaining the training data uses three external sources: a set of transport-layer flows, the assignments between the application name and traffic classes, and the assignments between the content types and traffic classes. The origin of the mentioned data sources is described below.

A. Transport-layer flows

The transport-layer flows are obtained by our Volunteer-Based System (VBS), whose architecture and implementation was described in [2]. We implemented several major changes to the system since it was published in order to make it capable to process the information about HTTP content types.

First, we used the JNetPcap library to detect the HTTP header in each packet of the flow and, in case of presence, to extract the values of the *content-type* field. The field is always present in the first incoming packet of the logical HTTP flow inside the transport-layer flow. Obtaining this information allows us not only to detect the class of the traffic but also to separate logical HTTP flows within one transport-layer flow. The extracted values of the *content-type* field are associated with particular packets and sent to the server where they are stored in a separate table and associated with the corresponding packets. This way, we are able to keep track of the content types in one place and save space. Due to obtaining all the relevant information directly from the client machines, VBS fulfills two roles. First, it is an independent distributed solution able to classify network traffic in real-time from the machine where it is installed. Second, it delivers the data for training the C5.0, which is used in the centralized classification approach. We must admit that turning on the HTTP header inspection did not increase the CPU usage in a measurable way, so there was no need to implement any optimization methods (like processing only a part of the flows, inspecting only a a part of the packets in a flow, and so on).

B. Assignments between the application names and traffic classes

Obtaining the mappings between application names and traffic classes is quite straightforward and it was done in the way described below. The results for our case are shown in Table II. It is worth mentioning that one element can possibly match multiple classes. For example, a *p2p* flow can carry a file (so it can be in fact a good match for the *file download* class). The *audio* and *video* classes carry in this case only the streamed content. The *http* process in a standard Ubuntu application, which is responsible for downloading files for system purposes, as system upgrades.

- Extract all the application names from the flows, which contain at least 5000 packets in total. This limitation was made to prevent including in the listing applications which generated only small amounts of data, because they are not sufficiently representative.
- Change all the names to lowercase and trim the whitespace from both ends. Then write the list to a Comma Separated Values (CSV) file.

Table II
MAPPING APPLICATION NAMES TO TRAFFIC CLASSES

Name	Requirement	Class
amule		p2p
dropbox		file download
filezilla		file download
http		file download
java		file download
libgcflashplay	remote_port = 1935	video
plugin-container	remote_port = 1935	video
skype	protocol_name = 'UDP'	audio
ssh		ssh
steam		file download
utorrent		p2p
wget		file download

- Manually assign a traffic class to all the rows in the file. Add a condition as a part of an SQL statement, if needed (for example by restricting the transport layer protocol to UDP or including only flows matching particular port numbers). Some of the applications also can generate background traffic, which must be cut off (like Skype, which beside the main voice UDP flow generates numerous TCP connections to other clients to exchange the distributed users' directory). If the application is unknown or it can handle different kinds of traffic which cannot be separated by a SQL statement, it should be deleted from the list.

C. Assignments between the content types and traffic classes

To be able to map the logical HTTP flows to a traffic class, we needed to create a mapping table based on the information contained in the database. This resulted in the assignments shown in Table III. The process of obtaining these mappings is described below. It is worth mentioning that one element can possibly match multiple classes, as the classes present various levels: content (as *audio*) or behavior (as *file download*). For example, an *audio* flow can carry a streamed content from a web radio or an audio file (so it can be in fact a good match for the *file download* class).

- Extract all the values of the content types from the packets, changing their names to lowercase and trimming the whitespace from both ends.
- Remove from the content type everything beyond the type itself, for example, the information about the used encoding. Then, write the list to a CSV file.
- Manually assign a traffic class to all the rows in the file. If the content type cannot be verified, delete the row. If the content-type can correspond both to the audio and to the video traffic, assign the *multimedia* class.

Unfortunately, relying on mappings between the traffic classes and the content types is not always accurate and consistent regarding the QoS assessment. For example, a movie downloaded (directly by the user from a website or indirectly in the background by the browser from YouTube) is marked as the *video* flow, as it carries video content. However, regarding the QoS requirements, the *video* class should contain only streamed video content, and the most appropriate action in this case is to mark the flow as the *file download*, but there is no simple way to obtain knowledge about the purpose of the traffic beside asking the user what he is currently doing.

Table III
MAPPING CONTENT TYPES TO TRAFFIC CLASSES

Class	Content type
audio	audio/aac, audio/aacp, audio/mpeg, audio/x-mpegurl, audio/x-pn-realaudio-plugin, audio/x-scpls
file download	application/binary, application/force-download, application/octet-stream, application/octetstream, application/pdf, application/rar, application/x-bzip2, application/x-compress, application/x-debian-package, application/x-gzip, application/x-msdos-program, application/x-msdownload, application/x-redhat-package-manager, application/x-tar, application/x-xpinstall, application/x-zip-compressed, application/zip, binary/octet-stream
multimedia	application/x-mms-framed, application/ogg
video	application/x-fcs, flv-application/octet-stream, video/mp4, video/ogg, video/webm, video/x-flv, video/x-m4v, video/x-ms-asf, video/x-msvideo
web	application/atom+xml, application/gif, application/java-archive, application/javascript, application/js, application/json, application/ocsp-request, application/ocsp-response, application/opensearchdescription+xml, application/pkix-crl, application/rdf+xml, application/rss+xml, application/smil, application/soap+xml, application/x-amf, application/x-director, application/x-font, application/x-httpd-cgi, application/x-java, application/x-java-archive, application/x-javasc, application/x-javascri, application/x-javascrip, application/x-javascript, application/x-ns-proxy-autoconfig, application/x-pkcs7-crl, application/x-sdch-dictionary, application/x-shockwave-flash, application/x-silverlight-app, application/x-woff, application/x-ww, application/x-www, application/x-x509-ca-cert, application/xaml+xml, application/xhtml+xml, application/xml, banner/jpg, font/woff, httpd/unix-directory, image/bmp, image/gif, image/ico, image/jpeg, image/jpg, image/pjpeg, image/png, image/svg+xml, image/vnd.microsoft.icon, image/x-ico, image/x-icon, image/x-ms-bmp, image/x-png, multipart/byteranges, multipart/form-data, text/css, text/html, text/javascript, text/json, text/plain, text/vdf, text/x-c, text/x-cross-domain-policy, text/x-gwt-rpc, text/x-javascript, text/x-js, text/x-json, text/x-perl, text/xml

IV. CLASSIFICATION BY C5.0

During the experiment, we used the same sets of classification attributes (A plus B) as used in [3]. We performed a normal decision-tree based classification for two cases: only for HTTP traffic, and for the mixed HTTP/non-HTTP traffic. In the first case, we tried to estimate the optimal values for the parameters M and N in the algorithm, so we made several tries with various values of M and N while observing how it affects the classification error. Both parameters equal to 0 mean that the mechanism of switching flows between the *web* and the *file download* classes is turned off. The matrix of the classification error for HTTP flows while using different values of M and N is shown in Table IV.

As shown, the accuracy of the classifier is independent of the lower and upper limits (M and N) for the interactive web traffic. Moreover, we can turn off the changing-class mechanism without significant decrease of the accuracy. We observe this behavior, because in order to test the classification accuracy, we use a disjoint set of data obtained in the same way as the set used for the training purposes, so it uses the

Table IV
CLASSIFICATION ERROR RATE [%] FOR DIFFERENT VALUES OF M AND N

	N=0	N=100	N=200	N=300	N=500	N=1000
M=0	17.9					
M=30		17.3	17.2	17.2	17.2	17.2
M=60		17.1	17.2	17.2	17.2	17.2
M=100		17.1	17.3	17.2	17.2	17.3
M=150			17.3	17.3	17.3	17.3
M=300				17.2	17.2	17.2
M=500					17.3	17.2

```
   (a)    (b)    (c)    (d)    <-classified as
 -----  -----  -----  -----
 98.04   0.40   1.03   0.53    (a): class audio
        78.08  20.96   0.95    (b): class file download
  0.02  12.67  85.95   1.37    (c): class video
  0.26  10.67   9.52  79.56    (d): class web
```

```
   (a)     (b)     (c)     (d)     (e)     (f)    <-classified as
------  ------  ------  ------  ------  ------
 95.89    0.64    1.37            1.72    0.38    (a): class audio
  0.01   86.03    0.72           12.51    0.73    (b): class file download
          0.13   99.85            0.01            (c): class p2p
          0.75           96.79    0.35    2.10    (d): class ssh
  0.02   12.85    0.06           86.12    0.95    (e): class video
  0.13   11.88    0.16    0.02    9.40   78.42    (f): class web
```

Figure 2. Misclassification matrix [%] for the HTTP traffic (above) and entire traffic (below)

same values of M and N. It means that the proper values for M and N should be estimated through observations of the traffic. It ensures that the *web* class contains as much of the real interactive browser traffic as possible, but as the least of the multimedia and the file transfer traffic.

For the HTTP traffic (the misclassification matrix is shown in Figure 2 (M=100, N=300)), we have two major observations. First, we are able to distinguish the *audio* and the interactive *web* traffic among different kinds of HTTP traffic. In our case, the *audio* group contained mostly web radios, which are of the streaming characteristic. Contrary to that, the *video* traffic and the regular *file download* transfers are often confused between themselves, as in fact, our *video* group contained in significant majority video files downloaded by a web browser, so their packet-level characteristic is the same as other file downloads. The average error rate was in this case 17.0 %.

During the second part of the experiment, we used full data sets containing both the HTTP and the non-HTTP traffic (the misclassification matrix is shown in Figure 2). For the non-HTTP traffic, we specified the following traffic classes: *audio*, *file download*, *p2p*, *ssh* and *video*. The non-HTTP video transfers were mostly streams played mostly through third-party plugins in the browser, such as Adobe Flash. The average error rate was in this case 6.0 %. The number of the misclassifications between the *file download* and the *video* classes decreased, as this time the *video* class contained more elements of the streaming characteristic than in the previous case.

V. TRAFFIC PROFILES

Based on the output from C5.0, we found the most used classification attributes to distinguish different types of the HTTP traffic. We chose two of them (the number of PSH flags for the inbound direction and the total payload size) to perform the graphical analysis. The distributions of these attributes shown in Figure 3 and in Figure 4 confirm that the *audio* and the *web* traffic differ significantly between each other, and from the *video* traffic and the big *file download* transfers. The number of the PSH flags increases when the content needs to be delivered to the client without delays. It proves that we can easily catch HTTP-based audio traffic, which is the most fragile for network performance issues. It justifies a need for the separate group of interactive *web* traffic as well. During this experiment, we used M=100 and N=300 in the algorithm generating the cases.

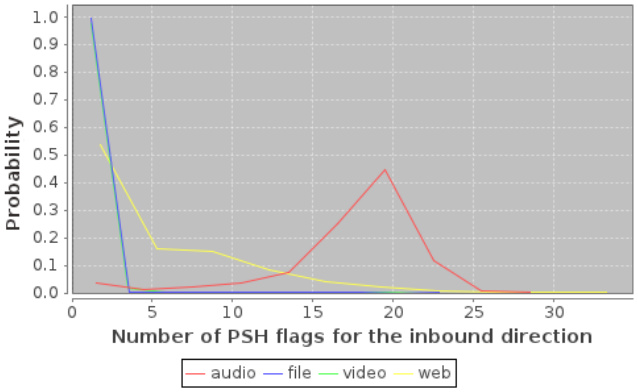

Figure 3. Distribution of number of PSH flags for the inbound direction

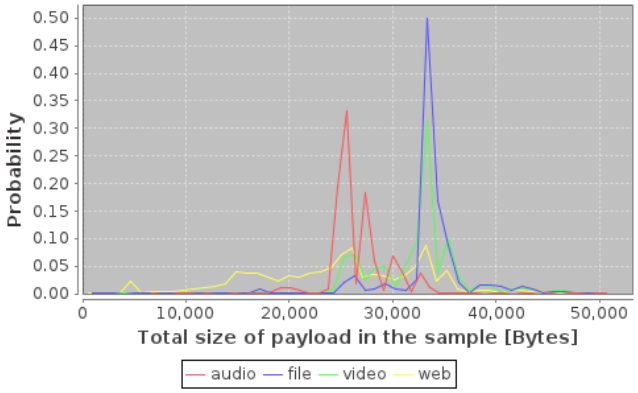

Figure 4. Distribution of total payload size in the sample

VI. CONCLUSION

This paper presents two novel methods for content-based recognizing different kinds of HTTP traffic in computer networks. The distributed method implemented among VBS clients uses the *content-type* fields in the HTTP headers to extract logical HTTP flows from the transport-layer flows. Later, the traffic classes are assigned based on the particular

types of the content. The centralized method is able to distinguish different content types transported by HTTP in the central point of the network based on the C5.0 MLA. We have shown that the MLA-based classifiers are not able to distinguish different types of the content transported by HTTP when the other flow characteristics (beside the content itself) are the same. The inability to distinguish between the video files and other binary files transported by HTTP caused the high average classification error rate (17.0 %). However, we demonstrated that the classifier did not have problems with recognizing interactive voice traffic, as it was originated mostly by streamed web radios. The last step of our experiment was to classify the mixed HTTP/non-HTTP traffic. In this case, we achieved much lower error rate of 6.0 %, as we included non-HTTP video streams from online TVs.

REFERENCES

[1] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008.

[2] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011.

[3] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012.

[4] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012.

[5] Gerhard Haßlinger. Implications of Traffic Characteristics on Quality of Service in Broadband Multi Service Networks. In *Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 196–204. IEEE, Rennes, France, September 2004.

[6] Kei Takeshita, Takeshi Kurosawa, Masayuki Tsujino, Motoi Iwashita, Masatsugu Ichino, and Naohisa Komatsu. Evaluation of HTTP video classification method using flow group information. In *Proceedings of the 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6. IEEE, Warsaw, Poland, September 2010.

[7] Samruay Kaoprakhon and Vasaka Visoottiviseth. Classification of audio and video traffic over HTTP protocol. In *Proceedings of the 9th International Symposium on Communications and Information Technology (ISCIT 2009)*, pages 1534–1539. IEEE, Icheon, Korea, September 2009.