Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification

Bujlow, Tomasz; Carela-Español, Valentín; Barlet-Ros, Pere

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification

Technical Report

from the project work made during January 7, 2013 – April 7, 2013

PhD Student: Tomasz Bujlow (tbujlow@ac.upc.edu)

PhD Student: Valentin Carela-Español (vcarela@ac.upc.edu)

Supervisor: Pere Barlet-Ros (pbarlet@ac.upc.edu)

Department of Computer Architecture (DAC)

Version 3: June 30, 2013

Previous versions:
Version 1: June 6, 2013
Version 2: June 28, 2013

Abstract

Nowadays, there are many tools, which are being able to classify the traffic in computer networks. Each of these tools claims to have certain accuracy, but it is a hard task to asses which tool is better, because they are tested on various datasets. Therefore, we made an approach to create a dataset, which can be used to test all the traffic classifiers. In order to do that, we used our system to collect the complete packets from the network interfaces. The packets are grouped into flows, and each flow is collected together with the process name taken from Windows / Linux sockets, so the researchers do not only have the full payloads, but also they are provided the information which application created the flow. Therefore, the dataset is useful for testing Deep Packet Inspection (DPI) tools, as well as statistical, and port-based classifiers. The dataset was created in a fully manual way, which ensures that all the time parameters inside the dataset are comparable with the parameters of the usual network data of the same type. The system for collecting of the data, as well as the dataset, are made available to the public. Afterwards, we compared the accuracy of classification on our dataset of PACE, OpenDPI, NDPI, Libprotoident, NBAR, four different variants of L7-filter, and a statistic-based tool developed at UPC. We performed a comprehensive evaluation of the classifiers on different levels of granularity: application level, content level, and service provider level. We found out that the best performing classifier on our dataset is PACE. From the non-commercial tools, NDPI and Libprotoident provided the most accurate results, while the worst accuracy we obtained from all 4 versions of L7-filter.

Contents

Chapter 1

Introduction

Classification of traffic in computer networks is a very challenging task. Many different types of tools were developed for that purpose. The first generation of tools used port-based classification [1, 2]. This fast technique is supported on most platforms, but its accuracy decreased dramatically during time, because of increasing share of protocols, which use dynamic port numbers. This concern especially Peer-to-Peer (P2P) applications, as eMule or BitTorrent [3, 4, 5]. Furthermore, some of applications on purpose use different port numbers than the standard one – this approach allow them to cheat port-based classifiers and obtain higher bandwidth, or higher priority in the network.

Because of the drawbacks of the port-based tools, a new technique called Deep Packet Inspection (DPI) was introduced. Because it relies on inspecting of the real payload [6], it is not possible to cheat the classifier by using non-standard port numbers. Apart from this big advantage, DPI also has many drawbacks. First of all, it cannot be used in many countries because of the local law. Second, even, if it is legal, it is often not used due to many privacy issues [3]. Third, it requires significant amount of processing power [3, 4]. Finally, in some cases DPI is not possible because of used encryption techniques, or because the application or protocol changed its signature [3].

The third generation of network classification tools are statistical-based tools, which use various Machine Learning Algorithms (MLAs). They do not inspect the payload, but they rely on the behavior of the traffic (as packet sizes and their distribution, or time-based parameters). Sometimes other network or transport layer parameters are also included, as port numbers or DSCP. Because of this simplicity, MLAs can offer high accuracy compared to DPI tools (it is claimed to be over 95 %), while preserving low resource demands [1, 2, 3, 5, 6, 7, 8, 9]. Based on the ML technology (C5.0 algorithm) at UPC was developed a tool for classification of network traffic, which has

an accuracy of 88-97 % [10] in distinguishing of 14 different main application classes.

To test the accuracy of any classification tool, we need to have a set of data of a good quality. Some datasets are available to the public (for example Caida sets [11]). Unfortunately, they do not contain all the data – often they miss the real payload, transport layer information, IP addresses, or inter-arrival times of the packets. Thanks to that, their usefulness in the development and testing of the classification tools is limited. Moreover, the datasets are already pre-classified by some tools; either port-based tools, or DPI tools. Even if they contain the original payload, we are not able to build the testing dataset based on the provided sets, because in order to do that, we would need to pre-classify them by some other classification tool.

To overcome that problem, we decided to build the dataset used for testing by ourselves. For this purpose we used a tool developed at Aalborg University, called *Volunteer-Based System (VBS)*. Windows, Linux, and source versions of this tool were published under *GNU General Public License v3.0* and they are available as a SourceForge project [12]. The task of the project is to collect flows of Internet traffic data together with detailed information about each packet. For each flow we also collect the process name associated with it from the system sockets. Additionally, the system collects some information about types of transferred HTTP contents. The design of the *Volunteer-Based System* was initially described in [13]. Further improvements and refinements can be found in [14]. We decided to use the system, since it was successfully used in many previous approaches [15, 16, 17, 18, 19]. The original *Volunteer-Based System* was modified by us in order to collect additionally the complete packets and some other information useful for data analysis.

In this paper we focus on two main tasks. The first task is to build a dataset, which will be useful for the testing purposes. The built dataset consists of PCAP files, which contain the real packets ordered by their timestamp and the information files, which describe each flow in details. The flow start and end time is provided, the process name associated with that flow, and some information which were extracted to make the analysis easier (as IP addresses, ports, associated types of HTTP content, etc). The dataset will be available to the public, so that other researchers can test their classifiers and compare their accuracy to the results obtained by us. The second part of the paper focuses on testing different DPI tools. For that purpose, we chose Ipoque's Protocol and Application Classification Engine (PACE), OpenDPI, L7-filter, NDPI, Libprotoident, and Cisco NBAR. Table 1.1 summarizes these DPI-based tools along and their characteristics. We also demonstrate how our own solution, developed at UPC, performs comparing to these DPIs. Our

Table 1.1: DPI-based techniques evaluated

Name	Version	Applications
PACE	1.41 (June 2012)	1000
OpenDPI	1.3.0 (June 2011)	100
nDPI	rev. 6391 (March 2013)	170
L7-filter	2009.05.28 (May 2009)	110
Libprotoident	2.0.6 (Nov 2012)	250
NBAR	15.2(4)M2 (Nov 2012)	85

tool is based on C5.0 Machine Learning Algorithm and it does not perform any DPI, but it is trained by data pre-classified by PACE. While testing the performance of different classification tools, we took into account three main parameters: accuracy, coverage (what amount of cases were left unclassified), and granularity (how detailed the classification is).

The remainder of this paper is structured as follows. We start by describing how we build the dataset used for testing in Chapter 2. In Chapter 3, at first we present the overview of our methods, then we show our working environment and the equipment used by us (Section 3.1), and describe how me modified the *Volunteer-Based System* in order to adjust it to match our needs (Section 3.2). Afterwards, in Section 3.3 we show how the data was extracted to be processed by different classification tools. Section 3.4 describes how the data is processed by the classifiers and Section 3.5 shows how the classification logs are processed to match the classification results to the flows stored in the database. In Chapter 4 the obtained results are shown and discussed. Chapter 5 finalizes the paper.

Chapter 2

The dataset

Our first task was to create a dataset, which will be used for the testing purposes. To create a representative dataset we decided to combine the data on a multidimensional level. The particular levels will be described in the sections below.

2.1 Operating systems

Based on the statistics from web usage[1] we found that most PC users run Windows 7 (55.3 % of all users), Windows XP (19.9 % of all users), and Linux (4.8 %) - state for January 2013. Apple computers contribute for 9.3 % of the overall traffic, and mobile devices for 2.2 %. Because of the lack of the equipment as well as the necessary software for Apple computers and mobile devices, we decided to to create 3 virtual machines, which will cover 80.0 % of the used operating systems.

Each of our actions were performed using each of the following operating systems:

- Windows 7
- Windows XP
- Lubuntu (Ubuntu with LXDE - Lightweight X11 Desktop Environment)

[1] `http://www.w3schools.com/browsers/browsers_os.asp`

2.2 Applications

We decided to include in our research different types of traffic generated by various applications.

2.2.1 Web traffic

The most popular web browsers are[2]: Chrome (48.4 % of all users), Firefox (30.2 % of all users), and Internet Explorer (14.3 % of all users) - state for January 2013.

The web browsers included in the experiment were:

- Google Chrome (Windows 7, Windows XP, Linux)
- Mozilla Firefox (Windows 7, Windows XP, Linux)
- MS Internet Explorer (Windows 7, Windows XP)

2.2.2 Peer-to-peer traffic

We identified the most commonly used P2P sharing software based on the CNET Download ranking[3].

We found the software within both *Top downloads* and *Most downloaded last week* categories and we classified it according to the used P2P protocol. Gnutella clients (Ares Galaxy and iMesh) were not considered as it was impossible to find any popular legal files to be downloaded by these file sharing clients.

The P2P clients included in the experiments were:

A. Torrent protocol clients:
   - uTorrent (Windows 7, Windows XP)
   - Bittorrent (Windows 7, Windows XP)
   - Frostwire (Windows 7, Windows XP, Linux)
   - Vuze [old name: Azureus] (Windows 7, Windows XP, Linux)

B. eDonkey protocol clients:
   - eMule (Windows 7, Windows XP)
   - aMule (Linux)

[2] `http://www.w3schools.com/browsers/browsers_stats.asp`

[3] `http://download.cnet.com/windows/p2p-file-sharing-software/?tag=nav`

2.2.3 FTP traffic

We identified the most commonly used FTP software based on the CNET Download ranking[4].

The FTP clients included in the experiment were:

- FileZilla (Windows 7, Windows XP, Linux)
- SmartFTP Client (Windows 7, Windows XP)
- CuteFTP (Windows 7, Windows XP)
- WinSCP (Windows 7, Windows XP)

2.2.4 Remote Desktop traffic

Most of the operations made on the machines included in the experiment were performed using Remote Desktop connections. On each of them we needed to install (or enable) Remote Desktop servers.

The Remote Desktop servers included in the experiment were:

- built-in Windows Remote Desktop server (Windows 7, Windows XP)
- xrdp (Linux)

2.2.5 SSH traffic

Some operations on Linux machines were made by SSH. The only SSH server included in the experiment was:

- sshd (Linux)

2.2.6 Background traffic

Our dataset also contains a big share of background traffic. We identified and included in our research the following applications:

- DNS traffic (Windows 7, Windows XP, Linux)
- NTP traffic (Windows 7, Windows XP, Linux)
- NETBIOS traffic (Windows 7, Windows XP)

[4] `http://download.cnet.com/windows/ftp-software/?tag=mncol;sort&rpp=30&sort=popularity`

2.3 Operations / contents

For each of the applications described above we tried to present different types of behavior and deal with various kinds of contents.

2.3.1 Web traffic

The list of visited websites was based on the Alexa statistics[5].

We chose several websites from the top 500, based on their rank and on the character of the website (as search engines, social medias, national portals, video websites, etc) to assure the variety of produced traffic. We chose English versions of websites if it was possible. From majority of the websites we performed some random clicks to linked external websites, which should better characterize the real behavior of the real users. This also concerns search engines, from which we generated random clicks in the destination web sites. The methodology of making the random clicks is as follows:

a. Go to the website from which the random click is going to be made

b. Scroll the website to a random position (or do not scroll at all)

c. Click a random link in the visible are of the website

Each of the chosen by us websites was processed by each browser chosen by us for the experiment. In case if it was required to log into the website, we created fake accounts. The visited by us websites were:

A. Google - big Internet portal with integrated search engine and mail hosting (rank 1): `https://www.google.com`

B. Facebook - social media portal (rank 2): `http://www.facebook.com`

C. YouTube - video sharing portal (rank 3): `http://www.youtube.com`

D. Yahoo! - big Internet portal with integrated search engine and mail hosting (rank 4): `http://www.yahoo.com`

E. Wikipedia - a free encyclopedia (rank 5): `http://www.wikipedia.org`

F. Java - a portal for Java developers, offering downloads of JVMs and JDKs (rank 343): `http://java.com/en`

[5] `http://www.alexa.com/topsites`

G. Justin.tv - live video streaming portal (rank 1227): `http://www.justin.tv`

The detailed description of actions performed with the services is listed below. The actions were designed by us, to match the biggest possible scope of the users' behavior.

A. Google

Logins and passwords for Google accounts:

- fake.user.upc / password
- fake.user.upc2 / password
- fake.user.upc3 / password

The operations performed on Google:

- Gmail:
  a. log into Gmail
  b. read 10 different e-mails
  c. send 2 e-mails without attachments
  d. send 1 e-mail with attached 6 pictures (around 2 MB each)
- the search engine - for each term from the top 10 searched terms on Google[6]:
  a. browse the first 10 search results. This should give us more realistic traffic in out set, since users tend to browse websites which are on the top of results from search engines
  b. browse Google Images associated with that term
  c. go to Google Maps and try to look for places associated with that term. Then, select one random place and zoom until the Street View appears. Afterwards, *turn around* until all the 360 degrees view from Street View is downloaded

B. Facebook

Logins and passwords for Facebook accounts:

- fake.user.upc@gmai.com / password
- fake.user.upc2@gmail.com / password

[6] `http://www.google.com/trends/explore`

- fake.user.upc3@gmail.com / password

The operations performed on Facebook:

- join some Facebook groups (1-5)
- post on the group
- like some posts on the group
- add some comments to someone's comments on the group
- invite some friends
- accept invitation from other friends
- browse pictures of Enrique Iglesias
- add some personal details to the profile
- like some pages (10-20)
- posts on a page which you like
- like some posts on a page which you like
- comment some posts on a page which you like
- share some photos from pages which you like
- attend few events
- invite friends to that events
- accept invitation for events from other friends
- share some events on the wall
- create an event
- invite friends for the event created by ourselves
- make some posts and likes on the page of our event
- post something on our wall
- like some posts on other people wall
- comment some posts on other people wall
- upload 29 pictures (60 MB)
- browse the pictures which we uploaded
- browse a page called *My Afghanistan Best At All*
- watch some videos on the page *My Afghanistan Best At All*

C. YouTube

The watched videos are the most watched videos from all the times[7]

Logins and passwords for Google accounts:

- fake.user.upc / password
- fake.user.upc2 / password
- fake.user.upc3 / password

The operations performed on YouTube:

- watch the 10 most popular videos (global ranking)
- make some comments
- click randomly *Like* or *Not like*
- try to pause some random videos from the list and then resume them
- try to rewind forward or backward some random videos from the list

D. Yahoo!

Logins and passwords for Yahoo! accounts:

- fake.userupc1@yahoo.com / password
- fake.userupc2@yahoo.com / password
- fake.userupc3@yahoo.com / password

The operations performed on Yahoo!:

- login to the service
- search for something, see various images, photo galleries and videos
- browse news, including videos and photo galleries
- autos
- games
- horoscopes
- jobs
- mail: read messages, sent messages/replies without attachment, send one message with few pictures attached

[7]Global ranking: `http://www.youtube.com/charts/videos_views?t=a`

- movies
- music
- shopping
- sports
- travel
- weather
- download few files from Yahoo Downloads

E. Wikipedia

   The watched sites are the 10 most searched terms in Wikipedia for each language[8]:

   - English
   - Dutch
   - German
   - Spanish
   - Japanese

F. Java

   Several big files (30 - 150 MB: Java JDKs) are downloaded in order to see how the big downloads will be classified.

G. Justin.tv

   Around 30 random short live video streams (1-10 minutes) were watched.

2.3.2 Peer-to-peer traffic

We tested the Torrent protocol clients by downloading few files of different size and then leaving the files to be seeded for some time in order to obtain enough of traffic in both directions. Peer-to-peer applications generate a big number flows per a file and, therefore, the number of files used in the experiment is sufficient. The links to the Torrent files were originated among the most common downloads from:

A. a website with legal torrents ClearBits[9]:

[8] `http://toolserver.org/~johang/2012.html`
[9] `http://www.clearbits.net/torrents/page/1/downloads`

- Megan Lisa Jones - Captive (BitTorrent Edition): 212 MB
- pearl-jam-life-wasted-video: 29.6 MB
- Sick of Sarah - 2205 BitTorrent Edition: 49.2 MB

B. the official Ubuntu website:

- ubuntu-12.10-desktop-amd64.iso: 763 MB

The eMule protocol clients were tested on 2 large files, which were every time searched in the internal search engine of each eMule protocol client:

- kubuntu-12.04.1-desktop-i386.iso: 703.29 MB
- kubuntu-12.10-desktop-amd64.iso: 934 MB

2.3.3 FTP traffic

We tried to test every FTP client using both the active transfer mode (PORT) and passive transfer mode (PASV). However, not all the clients support both modes and, therefore, the clients were tested in the following way:

- FileZilla: PORT + PASV
- SmartFTP Client: PORT + PASV
- CuteFTP: PORT + PASV
- WinSCP: only PASV

The following operations were performed by each FTP client (using all possible transfer modes):

- upload one directory with 29 pictures (60 MB)
- upload one big ZIP file (60 MB)
- browse the directory tree
- download again the directory with 29 pictures (60 MB)
- delete the directory from the server
- download again the big ZIP file (60 MB)
- delete the big ZIP file from the server

Logins and passwords for Yahoo! accounts:

- server: ftp.wlan.webd.pl
- login: fake@wlan.webd.pl
- password: Elgayego0

Chapter 3

Methodology

Testing different network traffic classifiers involved a number of various tasks. At first, the dataset used for testing had to be build. That required installing necessary machines in desired configurations (operating systems, applications, etc) and equipping them in a data collecting software. To collect the traffic we decided to use a modified version of the Volunteer-Based System developed at Aalborg University. Thanks to it we could collect all the packets passing the network interfaces, where the packets are grouped into flows, and the process name taken from the system sockets is assigned to each flow.

When the data were collected into a central database, they needed to be properly extracted into a format, which will be understood by the classifiers. Therefore, a new component of the Volunteer-Based System, called *pcapBuilder*, was developed. After processing the input data, the DPI tools generate log files, which needed to be imported back into the database to analyze accuracy of the classification. To deal with the various types of log files, a new component of the Volunteer-Based System, called *logAnalyzer*, was developed. The analyzed results were shown later in this paper.

3.1 The equipment

We decided to use 4 virtual machines (VMware) - 3 for each client, and 1 for the server. The detailed configuration is described below. All remote desktop connections presented here are established in the full-screen mode. To switch between full-screen and non-full screen modes, use *Left Ctrl + Left Alt + Enter*.

A. Lubuntu Virtual Machine - VBS Client

    1. Credentials (user/password): login/password

- Remote desktop - LXDE:
  rdesktop VBS_Ubuntu.pc.ac.upc.edu -u login -x l -f
- Terminal with X functionality:
  ssh -X login@VBS_Ubuntu.pc.ac.upc.edu

2. Credentials (user/password): login/password
   - Remote desktop - LXDE:
     rdesktop VBS_Ubuntu.pc.ac.upc.edu -u login -x l -f
   - Terminal with X functionality:
     ssh -X login@VBS_Ubuntu.pc.ac.upc.edu

B. Windows XP 32bit Virtual Machine - VBS Client

1. Credentials (user/password): login/password
   - Remote desktop:
     rdesktop VBS_XP32.pc.ac.upc.edu -u login -x l -f
   - To change the keyboard language (Spanish/English) use "-k es" or "-k en".

C. Windows 7 64bit Virtual Machine - VBS Client

1. Credentials (user/password): login/password
   - Remote desktop:
     rdesktop VBS_W764.pc.ac.upc.edu -u login -x l -f
   - To change the keyboard language (Spanish/English) use "-k es" or "-k en".

D. Ubuntu Virtual Machine - VBS Server for our virtual VBS clients

1. Credentials (user/password): login/password
   - Remote desktop - KDE:
     rdesktop classifier.cba.upc.edu -u login -x l -f
   - Terminal with X functionality:
     ssh -X -p 13000 login@classifier.cba.upc.edu
2. Credentials (user/password): login/password
   - Remote desktop - KDE:
     rdesktop classifier.cba.upc.edu -u login -x l -f
   - Terminal with X functionality:
     ssh -X -p 13000 login@classifier.cba.upc.edu

- To change the firewall rules, edit */etc/iptables/rules.v4*. After that, restart the firewall:
  /etc/init.d/iptables-persistent restart.
- To transfer a local file to the server use:
  scp -P 13000 /login/export/file.jar login@classifier.cba.upc.edu:
  /opt/directory/.
- To transfer a file from the server to a local disk use:
  scp -P 13000 login@classifier.cba.upc.edu:/opt/directory/traces.pcap
  /login/export/.

3.2 The system for collecting the data

On every virtual machine we installed a modified version of Aalborg University Volunteer-Based System for Research on the Internet. The source code of the original system as well as the modified version was published under *GNU General Public License v3.0* and it is available in GIT repository in the SourceForge project [12]. The modified version of the system differs from the original one by several things:

- The client saves full captured frames as payloads.
- Each packet with an HTTP header is stored together with the corresponding URL and referrer.
- The server stores the payloads and the new information in the database.
- The client does not intercept the communication between the client and the server to prevent intercepting the traffic generated by itself.
- We increased the limit of the size of the database on the client side when the database is sent to the server.
- We decreased the size of the flow / number of packets in the memory before the packets are dumped to the local database.
- We changed the IP address in the client configuration file in order to make the connection from the new clients to the new server.
- The server has increased RAM availability in the YAJSW config file.
- The IP addresses are stored in non-hashed version in the database.
- The performance statistics are not generated.

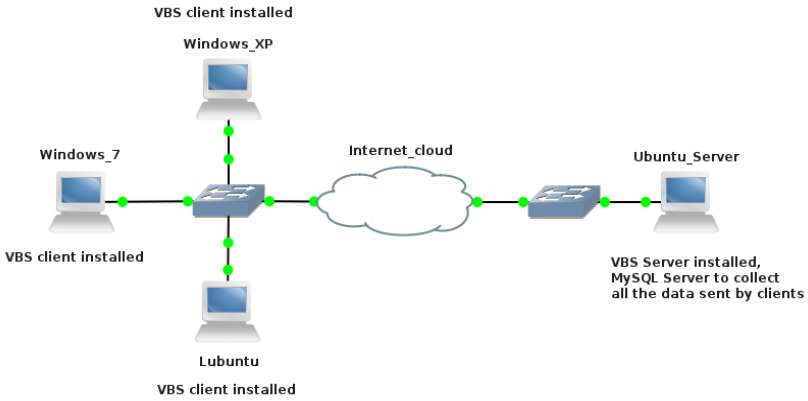

Figure 3.1: Topology of our testbed

- The real timestamps are stored instead of relative timestamps to make easier ordering of the packets.
- Provider network names are not supported.
- We added a module called *pcapBuilder*, which is responsible for dumping all the flows to PCAP files. At the same time, INFO files are generated to provide detailed information about each flow, which allows to assign each packet in the PCAP file to the individual flow.
- We added a module called *logAnalyzer*, which is responsible for analyzing logs generated by different DPI tools, and assigning the results of the classification to the flows in the database.

The topology of our virtual machines with the installed components of VBS is shown in Figure 3.1.

3.3 Extracting the data for processing

The data stored in the database must be extracted in a format which will be handled by the DPI tools and our Machine Learning tool. Each tool can have different requirements and possibilities, so the extracting tool must handle all these issues. The most challenging thing is instructing the software, which handle the DPIs and the ML tool, how to construct flows in the identical way as they were constructed by the Volunteer-Based System. The biggest problem is that VBS opens and closes flows based on opening or closing of the system sockets - the DPI and ML tools do not have such information.

So, the additional information about the start and end of each flow must be provided to the classifiers as well.

Extracting all flows from the database is done separately for each client from which the data was collected. Thanks to that, it is easier to share the files between different researchers, or to make a small-scope classification on a limited amount of data. The data is extracted into PCAP files, which contain all packets ordered according to their absolute timestamps, so that the packets are provided to the classifiers in the original order. Some classifiers can rely on the flow coexistence feature (many flows from the same IP address), or use DNS requests to obtain the names of particular services. Therefore, we cannot provide the packets to the classifier ordered by flows – these would influence the flow coexistence.

Extracting of the packets can be automatized by our *pcapBuilder* tool (which is a part of the modified VBS system). At first, it is advised to check the database for consistency and repair it in case of any problems. Such problems can arise if, for example, the capture of the packets was interrupted by system restart or hangs of the capturer itself. Checking of the consistency (and repairing the database if needed) can be done by:

```
pcapBuilder --fixDatabaseStructure
```

Later, we can start generating the input for the desired classification tools.

3.3.1 PACE, OpenDPI, L7-filter, NDPI, and Libprotoident

To generate the input for PACE, OpenDPI, L7-filter, NDPI, and Libprotoident we can use the following syntax of our *pcapBuilder* tool:

```
pcapBuilder --writeDefault [all | infos] [express | normal]
   [0=allClients | client_id]
```

depending if we want to:

a. generate PCAP and INFO files [*all*], or only INFO files [*infos*]

b. generate only basic INFO files without application names, urls, referrers, and content type information [*express*] or full set of data [*normal*]

c. generate the files only for selected client [*client_id*] or for all clients [*0*]

For each client a set of 2 files is generated:

- a PCAP file [*packets_all_X.pcap*], which contains all the flows. The packets are ordered by their absolute timestamps
- an INFO file [*packets_all_X.info*], which contains the description of the flows. Based on the description the classifiers are able to construct the flows in the same manner as they were constructed by our system

The format of each row in the INFO file is as follows:

```
flow_id + "#" + start_time + "#" + end_time + "#" + local_ip + "#"
   + remote_ip + "#" + local_port + "#" + remote_port + "#"
   + transport_protocol + "#" + operating_system + "#"
   + process_name + "#" + urls + "#" + referrers + "#"
   + content_types + "#"
```

3.3.2 NBAR

At first, we need to extract the packets in a way which will allow them to be processed by the router and to be correctly grouped into flows. We achieve that by changing both source and destination MAC addresses during the extraction process. The destination MAC address of every packet must match up with the MAC address of the interface of the router (set by us to be *ca:00:11:5b:00:00*). The router cannot process any packet which is not directed to its interface on the MAC layer. The source MAC address is set up to contain the identifier of the flow to which it belongs. Cisco router (and NBAR) does not have any knowledge of when a flow starts or ends. The default behavior is to impose a timeout, but it does not work in our case because of two things:

- Flows stored by our system are not closed based on timeout, so the flows generated by NBAR would not be matched 1:1 to the flows in our system
- We are replaying the packets to the Cisco router with the maximal speed which does not cause packet drops. It means that in many cases the timeout will not appear and many flows of the same 5-tuple would be just merged together.

To generate the PCAP files for NBAR, our *pcapBuilder* tool can be used - it required the destination MAC address of the Cisco router as a parameter:

```
java -jar pcapBuilder.jar --writeNBAR ca:00:11:5b:00:00
```

3.3.3 UPC Machine Learning Classification Tool

We prepared the extractor to be able to deliver the data to test the classification tool developed at UPC, which is based on C5.0 Machine Learning Algorithm. For MLAs, all the flows from each of the clients must be split into sets of training and test flows. It means that we will have a set of 2 PCAP files (and 2 INFO files), which will be used for training and testing MLAs. There is no reason to make separate sets for each client – the files produced in this step are not supposed to be shared, as everyone can produce them from the full set of data obtained in the previous step. Each flow will be randomly assigned to the training set or to the testing set. To generate the input for the MLA tool, we can use the following syntax of our *pcapBuilder* tool:

```
pcapBuilder --writeMLA [all | infos] [express | normal]
```

depending if we want to:

a. generate PCAP and INFO files [*all*], or only INFO files [*infos*]

b. generate only basic INFO files without application names, urls, referrers, and content type information [*express*] or full set of data [*normal*]

A set of 4 files is generated:

- a PCAP file [*packets_train_X.pcap*], which contains around half of the flows used for the training purposes. The packets are ordered by their absolute timestamps
- an INFO file [*packets_train_X.info*], which contains the description of the flows used for the training purposes. Based on the description the classifiers are able to construct the flows in the same manner as they were constructed by our system
- a PCAP file [*packets_test_X.pcap*], which contains around half of the flows used for the testing purposes. The packets are ordered by their absolute timestamps
- an INFO file [*packets_test_X.info*], which contains the description of the flows used for the testing purposes. Based on the description the classifiers are able to construct the flows in the same manner as they were constructed by our system

The format of each row in the INFO file is as follows:

```
flow_id + "#" + start_time + "#" + end_time + "#" + local_ip + "#"
   + remote_ip + "#" + local_port + "#" + remote_port + "#"
   + transport_protocol + "#" + operating_system + "#"
   + process_name + "#" + urls + "#" + referrers + "#"
   + content_types + "#"
```

3.4 The classification process

3.4.1 PACE, OpenDPI, and the version of L7-filter used for automatic retraining purposes

At UPC we designed a tool, called *dpi_benchmark*, which is able to read the PCAP files and provide the packets one-by-one to the relevant DPI classifiers. After the last packet of the flow is sent to the classifier, the tool is obtaining the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. To see all possible options of the classification we can run:

```
./dpi_benchmark -help
```

To process the set of PCAP and INFO files by the classifiers we execute the following command:

```
./dpi_benchmark -f path/to/pcap/file -b path/to/info/file >
   output/file
```

This operation is done separately for each client. The format of each row in the log files is:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#PACE_label#
   OpenDPI_label#L7_filter_label#ML_label1#ML_label2#
```

The MLA label will not be considered at this point, since MLA is not properly trained and the classifiers are tested on the full sets of data.
The label from L7-filter is originated from a modified by us version of L7-filter, which was used for our automatic retraining mechanism [10]. This

version does not have activated the patterns declared as *overmatching* and it has some patterns manually made by us to match the traffic from YouTube, Twitter, and Facebook. The priorities given to our patterns allowed to classify by these patterns the biggest possible amount of traffic.

3.4.2 L7-filter – the standard versions

The *dpi_benchmark* tool also has two versions, which support the standard versions of L7-filter. The first version has activated all the patterns, but the patterns marked as *overmatching* have low priority. The second version does not have activated the patterns declared as *overmatching*. The tools work as the tool described in the previous section – they read the PCAP files and provide the packets one-by-one to L7-filter. After the last packet of the flow is sent to the classifier, the tool is obtaining the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. To see all possible options of the classification we can run:

```
./dpi_benchmark -help
```

To process the set of PCAP and INFO files by the classifiers we execute the following command:

```
./dpi_benchmark -f path/to/pcap/file -b path/to/info/file >
   output/file
```

This operation is done separately for each client. The format of each row in the log files is:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#L7_filter_label#
```

3.4.3 L7-filter – Computer Networks version

At UPC, we also developed another version of L7-filter, which was used to process the biggest possible amount of traffic in the accurate way. The modifications for that version are described in our Computer Networks journal paper [20] and they rely on:

- The patterns are applied from the least overmatching to the most overmatching.
- Packets must agree with the rules given by pattern creators – otherwise the packet is not labeled.
- In case of multiple matches, the flow is labeled with the application given by the rule with the highest quality according to L7-filter documentation. In case if the quality of many patterns is equal, the first label matched is chosen.

The tools work as the tool described in the previous section – they read the PCAP files and provide the packets one-by-one to L7-filter. After the last packet of the flow is sent to the classifier, the tool is obtaining the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. To see all possible options of the classification we can run:

```
./dpi_benchmark -help
```

To process the set of PCAP and INFO files by the classifiers we execute the following command:

```
./dpi_benchmark -f path/to/pcap/file -b path/to/info/file >
   output/file
```

This operation is done separately for each client. The format of each row in the log files is:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#L7_filter_label#
```

3.4.4 NDPI

The *dpi_benchmark* tool also has a version, which supports NDPI classifier. It works as the tool described in the previous section – it reads the PCAP files and provides the packets one-by-one to NDPI. After the last packet of the flow is sent to the classifier, the tool is obtaining the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification

results to the original flows in the database. To see all possible options of the classification we can run:

```
./dpi_benchmark -help
```

To process the set of PCAP and INFO files by the classifiers we execute the following command:

```
./dpi_benchmark -f path/to/pcap/file -b path/to/info/file >
   output/file
```

This operation is done separately for each client. The format of each row in the log files is:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#NDPI_label#
```

3.4.5 Libprotoident

The *dpi_benchmark* tool also has a version, which supports Libprotoident. It works as the tool described in the previous section – it reads the PCAP files and provides the packets one-by-one to Libprotoident. After the last packet of the flow is sent to the classifier, the tool is obtaining the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. To see all possible options of the classification we can run:

```
./dpi_benchmark -help
```

To process the set of PCAP and INFO files by the classifiers we execute the following command:

```
./dpi_benchmark -f path/to/pcap/file -b path/to/info/file >
   output/file
```

This operation is done separately for each client. The format of each row in the log files is:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#Libprotoident_label#
```

3.4.6 NBAR

Choice of the proper NBAR version

There are 2 versions of NBAR in use: the casual NBAR and NBAR2. Unfortunately, NBAR2 is currently supported only on a very limited set of Cisco devices:

- Routers from 19xx, 29xx, and 39xx series[1]
- Other devices: ISR-G2, ASR1K, ASA-CX and Wireless LAN Controller[2]

So, the classification will be limited to the standard NBAR, which is still under constant development and which is included in most of Cisco devices and in the newest IOS from line 15.x.

Choice of the Cisco device and the operating system IOS

We did not have any free Cisco device which we can use for the experiment. However, we can use GNS3 - a graphical framework, which uses Dynamips to emulate Cisco hardware. The following Cisco platforms of routers can be emulated by Dynamips / GNS3:

- 1710, 1720, 1721, 1750, 1751, 1760
- 2610, 2611, 2610XM, 2620, 2620XM and 2650XM, 2611XM, 2621, 2621XM and 2651XM, 2691
- 3620, 3640, 3660
- 3725, 3745
- 7206

[1] Cisco Feature Navigator: `http://tools.cisco.com/ITDIT/CFN/`

[2] `http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6558/ps6616/qa_c67-697963.html`

```
NBAR - Citrix ICA Published Applications
NBAR - Multiple Matches Per Port
NBAR - Network-based Application Recognition
NBAR Extended Inspection for HTTP Traffic
NBAR PDLM Versioning
NBAR Real-time Transport Protocol Payload Classification
NBAR Static IPv4 IANA Protocols
NBAR User-Defined Custom Application Classification
NBAR- BitTorrent PDLM
NBAR-NAT Integration & RTSP
Flexible NetFlow
Flexible NetFlow - Ingress VRF Support
Flexible NetFlow - Output Features on Data Export
Flexible NetFlow: 32 bit AS Number Support
Flexible Netflow - IPv4 Multicast Statistics Support
Flexible Netflow - Layer 2 Fields
Flexible Netflow - MPLS Egress NetFlow
Flexible Netflow - NBAR Application Recognition
Flexible Netflow - NetflowV5 export protocol
Flexible Netflow - Top N Talkers Support
```

Figure 3.2: The interesting features contained by the IOS image

We chose 7200 platform, since only for this platform there is available the newest version of Cisco IOS (version 15), which contains Flexible NetFlow. Previous versions of Cisco IOS contain only traditional NetFlow, which do not support NBAR reporting on per flow basis. According to the Cisco Feature Navigator, the newest IOS for the 7200 platform, which contains interesting to us features, is:

```
Release:  15.2(4)M2
Platform:  7200
Feature set:  ADVANCED ENTERPRISE SERVICES
DRAM: 512
Flash:  64
Image:  c7200-adventerprisek9-mz.152-4.M2.bin
```

The set of the interesting features contained by the image is shown in Figure 3.2. We downloaded the IOS image from one of our routers, which are used in production, and used the image with GNS3. The router identifies itself as *Cisco IOS Software, 7200 Software (C7200-ADVENTERPRISEK9-M), Version 15.2(4)M2, RELEASE SOFTWARE (fc2)* – for the full listing see Figure 3.3.

```
Cisco IOS Software, 7200 Software (C7200-ADVENTERPRISEK9-M), Version 15.2(4)
    M2, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2012 by Cisco Systems, Inc.
Compiled Wed 07-Nov-12 18:15 by prod_rel_team

ROM: ROMMON Emulation Microcode
BOOTLDR: 7200 Software (C7200-ADVENTERPRISEK9-M), Version 15.2(4)M2, RELEASE
     SOFTWARE (fc2)

ROUTER0 uptime is 8 hours, 56 minutes
System returned to ROM by unknown reload cause - suspect boot_data[
    BOOT_COUNT] 0x0, BOOT_COUNT 0, BOOTDATA 19
System image file is "tftp://255.255.255.255/unknown"
Last reload reason: unknown reload cause - suspect boot_data[BOOT_COUNT] 0x0
    , BOOT_COUNT 0, BOOTDATA 19

This product contains cryptographic features and is subject to United States
...

If you require further assistance please contact us by sending email to
export@cisco.com.

Cisco 7206VXR (NPE400) processor (revision A) with 245760K/16384K bytes of
    memory.
Processor board ID 4279256517
R7000 CPU at 150MHz, Implementation 39, Rev 2.1, 256KB L2 Cache
6 slot VXR midplane, Version 2.1

Last reset from power-on

PCI bus mb0_mb1 (Slots 0, 1, 3 and 5) has a capacity of 600 bandwidth points
Current configuration on bus mb0_mb1 has a total of 200 bandwidth points.
This configuration is within the PCI bus capacity and is supported.

PCI bus mb2 (Slots 2, 4, 6) has a capacity of 600 bandwidth points.
Current configuration on bus mb2 has a total of 0 bandwidth points
This configuration is within the PCI bus capacity and is supported.

Please refer to the following document "Cisco 7200 Series Port Adaptor
Hardware Configuration Guidelines" on Cisco.com <http://www.cisco.com>
for c7200 bandwidth points oversubscription and usage guidelines.

1 FastEthernet interface
125K bytes of NVRAM.

65536K bytes of ATA PCMCIA card at slot 0 (Sector size 512 bytes).
8192K bytes of Flash internal SIMM (Sector size 256K).

Configuration register is 0x2102
```

Figure 3.3: The identification of the router by *show version* command

Connection of the router to the real network

In order to connect the router to the real network we needed to create a virtual interface on Linux (*tap0*) and bridge it creating a new virtual bridge

interface (*br0*). First of all, it is worth to highlight that the way of connecting the device to the computer without using bridge (but only *tap0* interface) does not work. Such way is described on many websites, but it is evidently just a copy-paste without checking if such approach works or not. At first, we create a virtual *Internet cloud* interface *tap0* and we bridge it to *br0*. Then, we set all the parameters of the bridge, as the IP address:

```
modprobe tun
tunctl -t tap0
ifconfig tap0 0.0.0.0 promisc up
brctl addbr br0
brctl addif br0 tap0
ifconfig br0 10.0.0.2 netmask 255.255.255.0 up
ifconfig tap0 mtu 65521
ifconfig br0 mtu 65521
```

If necessary, we can add any other interface to the bridge - for example to connect the router to the Internet:

```
ifconfig eth0 0.0.0.0 promisc up
brctl addif br0 eth0
```

or to connect to a VMWare virtual machine:

```
ifconfig vmnet1 0.0.0.0 promisc up
brctl addif br0 vmnet1
```

Deleting the interfaces is going in the opposite way:

```
brctl delif br0 tap0
ifconfig br0 down
brctl delbr br0
tunctl -d tap0
```

On the router side, we connect the *Fastethernet0/0* interface to the *tap0* interface of the Internet cloud.

Configuration of the router

We configure the router to enable Flexible NetFlow with NBAR on the *Fastethernet0/0* interface. NetFlow records will be sent back to the Linux

machine, where they will be stored and processed later. We also set a static MAC address on the interface, since every time the router is started, the "physical" MAC address is different. To connect to the router, we use telnet:

```
tomasz@kubuntu: $ telnet localhost 2001
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connected to Dynamips VM "R1" (ID 0, type c7200) - Console port
Press ENTER to get the prompt.
```

Now, we are going to present how the router was configured. The particular steps were shown and discussed in the following points:

1. General configuration

   We want to setup the router name to be more friendly. The router should by default discard all the packets which enter the interface, without further routing:

   ```
   hostname ROUTER0
   ip route 0.0.0.0 0.0.0.0 Null0
   ```

2. Configuration of the flow record

   The router must be instructed how to group the packets into flows (the *match* command), and which information for each flow should be collected (the *collect* command):

   ```
   flow record nbar-appmon
   --> description NBAR flow monitor
   --> match ipv4 protocol
   --> match ipv4 source address
   --> match ipv4 destination address
   --> match transport source-port
   --> match transport destination-port
   --> match datalink mac source address input
   --> collect counter bytes
   --> collect counter packets
   --> collect application name
   ```

3. Configuration of the flow exporter

The router must be instructed where the flows should be exported and which option tables should be included in the export. The option tables allows to match the identifiers of the NBAR classes to be matched to the actual names:

```
flow exporter export-to-computer
--> description flexible NF v9
--> destination 10.0.0.2
--> source FastEthernet0/0
--> transport udp 9995
--> template data timeout 60
--> option interface-table
--> option exporter-stats
--> option vrf-table
--> option application-table
--> option application-attributes
```

4. Configuration of the flow monitor

   The configured flow record must be associated with the configured flow exporter:

   ```
   flow monitor application-mon
   --> description app traffic analysis
   --> exporter export-to-computer
   --> cache timeout active 60
   --> record nbar-appmon
   ```

5. Configuration of the interface

   Every time GNS3 start, another MAC address is assigned to the interface. Because we need a fixed value (to be able to replay the packets to the interface), we assign a static one. Then, we need to enable NBAR on the interface and apply the created flow monitor:

   ```
   interface FastEthernet0/0
   --> mac-address ca00.115b.0000
   --> ip address 10.0.0.1 255.255.255.0
   --> ip nbar protocol-discovery
   --> ip flow monitor application-mon input
   --> duplex full
   ```

Configuration of the Linux computer

Computers running Linux can be tweaked to replay the packets to the network in an enhanced way. The following is known to apply to the 2.4.x and 2.6.x series of kernels. By default Linux's *tcpreplay* performance isn't all that stellar. However, with a simple tweak, relatively decent performance can be had on the right hardware. By default, Linux specifies a 64 K buffer for sending packets. Increasing this buffer to about half a megabyte does a good job:

```
echo 524287 >/proc/sys/net/core/wmem_default
echo 524287 >/proc/sys/net/core/wmem_max
echo 524287 >/proc/sys/net/core/rmem_max
echo 524287 >/proc/sys/net/core/rmem_default
```

Replaying the packets to the router

We can send the packets to the router using the bridge interface:

```
tcpreplay -i br0 --pps=3000 packets_nbar_1.pcap
tcpreplay -i br0 --pps=3000 packets_nbar_2.pcap
tcpreplay -i br0 --pps=3000 packets_nbar_3.pcap
```

To obtain the results separately for each user we need to setup the NetFlow analyzer on the computer separately for each PCAP file which is being replayed. It is worth mentioning that it is required to specify the speed with which the packets are replayed. Without specifying the speed, the packets would be replayed with the same speed as they were captured. It means that for our 2-months capture we would need to have the same 2-months replay period. The speed of replaying can be specified in packets per second or Megabytes per second. It is much better to use the first possibility, since the latter one cause enormous number of small packets sent during a short interval of time. This overloads the router and causes input queue drops. To adjust the number of packets per second which we are able to replay, we used the interface counter - no drops should be observed during the replay. To see that, we used *show interfaces* command of the router, and we observed the *input drops* parameter:

```
ROUTER0#show interfaces fastEthernet 0/0
FastEthernet0/0 is up, line protocol is up
...
```

```
ROUTER0#show flow monitor application-mon cache format table
  Cache type:                               Normal
  Cache size:                                 4096
  Current entries:                             140
  High Watermark:                              140

  Flows added:                                 381
  Flows aged:                                  241
    - Active timeout      (    60 secs)          0
    - Inactive timeout    (    15 secs)        241
    - Event aged                                 0
    - Watermark aged                             0
    - Emergency aged                             0

IPV4 SRC ADDR    IPV4 DST ADDR   SRCP  DSTP   PROT  app name
==============  ==============  =====  =====  ====  ==================
192.168.1.128   91.189.92.163   38126     80     6  port http
192.168.1.128   173.194.41.228  56856     80     6  cisco unclassified
173.194.41.228  192.168.1.128      80  56856     6  cisco unclassified
192.168.1.128   173.194.41.230  46235    443     6  port secure-http
173.194.41.230  192.168.1.128     443  46235     6  port secure-http
74.125.235.111  192.168.1.128      80  49617     6  cisco unclassified
192.168.1.128   91.189.90.143   56001   6969     6  cisco bittorrent
192.168.1.128   87.216.1.66         0    771     1  prot icmp
```

Figure 3.4: The flow monitor cache

```
Input queue:  0/75/0/0 (size/max/drops/flushes);
...
```

On the router side we can see that the flows are properly inspected and that the Flexible NetFlow entries are generated as expected. We can see that looking into the temporary cache of the router by *show flow monitor application-mon cache format table* (see Figure 3.4). To display the mappings between the application names and IDs we can use the *show flow exporter option application table* command (see Figure 3.5).

Receiving the NetFlow records by the Linux computer

There are many approaches to collect the NetFlow v9 records. Unfortunately, most of the tools which are supposed to work with NetFlow v9 do not support that format entirely. It means that either only some basic fields are supported, or the tools are not working at all if any custom field is added. This especially concerns field #95 - the application identifier. It took us around 3 weeks of experimenting with many different tools to find a one which is working properly with NetFlow v9 exports! Here there are some experiences with the tools (free and commercial) which are supposed to support NetFlow v9 format:

```
ROUTER0#show flow exporter option application table

Engine: prot (IANA_L3_STANDARD, ID: 1)

appID  Name                      Description
-----  ----                      -----------
1:8    egp                       Exterior Gateway Protocol
1:47   gre                       General Routing Encapsulation
1:1    icmp                      Internet Control Message Protocol
1:88   eigrp                     Enhanced Interior Gateway Routing Protocol
...

Engine: port (IANA_L4_STANDARD, ID: 3)

appID  Name                      Description
-----  ----                      -----------
3:21   ftp                       File Transfer Protocol
3:80   http                      World Wide Web traffic
3:179  bgp                       Border Gateway Protocol
...
3:25   smtp                      Simple Mail Transfer Protocol
3:53   dns                       Domain Name System

Engine: NBAR (NBAR_CUSTOM, ID: 6)

appID  Name                      Description
-----  ----                      -----------
6:244  custom-10                 Custom protocol custom-10
6:245  custom-09                 Custom protocol custom-09
...

Engine: cisco (CISCO_L7_GLOBAL, ID: 13)

appID  Name                      Description
-----  ----                      -----------
13:0    unclassified              Unclassified traffic
13:1    unknown                   Unknown application
13:9    ipsec                     IP Security Protocol (ESP/AH)
13:12   cuseeme                   CU-SeeMe desktop video conference
13:13   dhcp                      Dynamic Host Configuration Protocol
13:26   netbios                   Netbios
...
13:59   kazaa2                    Kazaa Version 2
13:554  rtsp                      Real Time Streaming Protocol
13:61   rtp                       Real Time Protocol
13:62   mgcp                      Media Gateway Control Protocol
13:63   skinny                    Skinny Call Control Protocol
13:64   h323                      H323 Protocol
13:66   rtcp                      Real Time Control Protocol
13:67   edonkey                   eDonkey
13:68   winmx                     WinMx file-sharing application
13:69   bittorrent                bittorrent
13:70   directconnect             Direct Connect Version 2.0
13:83   skype                     Skype Peer-to-Peer Internet Telephony
13:84   sap                       SAP Systems Applications Product in Data
...
```

Figure 3.5: Applications recognized by NBAR together with their IDs

A. NFDUMP

The standard tool NFDUMP[3] can collect only the standard fields from the NetFlow exports and it does not allow to collect any additional features, for example, the application name. However, we tested, if any NetFlow exports are collected at all. We started the capturing tool by:

```
nfcapd -p 9995 -b 10.0.0.2 -T all -t 61 -l /home/tomasz/nfcapd
```

Then, we processed the NetFlow records to obtain a human-readable version:

```
nfdump -o raw -R /home/tomasz/nfcapd
```

We confirmed that the NetFlow exports are correct, but as expected we did not obtain the application names. To see that everything is exported from the router as expected, we used the standard TCPDUMP tool:

```
tcpdump -i br0 -n 'src 10.0.0.1 and udp and dst port 9995'
   -w /home/tomasz/tcpdump.out
```

In case of problems, if any background process ix occupying a port and we need to know which process it is, it is sufficient to invoke the following command to obtain the application PID:

```
netstat -tulpn | grep 9995
```

where 9995 is the port number we want to inspect.

B. PMACCT

This set of PMACCT tools[4] is very powerful and it supports as well NetFlow v9 export format as field #95 (application name). However, to be able to capture flow records, all the records must contain packet counters and byte counters - without that, the flow records are ignored. We used the following command to obtain the relevant statistics:

[3] http://nfdump.sourceforge.net/

[4] http://www.pmacct.net/

```
nfacctd -L 10.0.0.2 -l 9995 -r 30 -c src_mac,src_host,dst_host,
   proto,src_port,dst_port,class -P print -O csv
   > nbar_results_X.txt
```

where X is the identifier of the client. There is also a possibility to put the configuration in a file:

```
nfacctd_ip:  10.0.0.2
nfacctd_port:  9995
plugins:  print[test]
!
aggregate[test]:  src_mac,src_host,dst_host,proto,src_port,
   dst_port,class
print_refresh_time[test]:  30
print_output[test]:  csv
print_output_file[test]:  /home/tomasz/nbar_results_X.txt
```

Afterwards, we can execute nfacctd as:

```
nfacctd -f nfacctd.cfg
```

For now it is not working properly, since every 30 seconds the file is completely overwritten with the new data (instead of just appending the data to the file) and there is no possibility to override this behavior.

This tool is the only tool tested by us which works with NetFlow v9 format including field #95 as it should! Therefore, we chose PMACCT to collect the NetFlow data from the router. The data collection process must be done in the following way:

- Start the NetFlow collector (*nfacctd*) on the computer
- Wait until at least one flow entry with other class than *unknown* appears in the log file. This is necessary since the collector must obtain from the router some special *option tables* before it will be able to recognize what is the application class. Before it happens, all the flows will be marked as *unknown*. No traffic generation is required during this step. Router generates multicasts by itself and they will be included in the log. This step can take even 10 minutes
- Replay the packets from the pcap file to the router

C. Scrutinizer

   Scrutinizer is supposed to be a tool which can not only collect, but also visualize the network traffic. It consumes a lot of resources, especially RAM (around 700 MB). Furthermore, it does not record the NetFlow v9 packets (but they are captured, because Wireshark can see them arriving).

D. ManageEngine NetFlow Analyzer

   In theory, a big flow analyzer, which is supposed to support NetFlow v9 record format, NBAR, etc. It can be downloaded for free from the developer's website[5]. Unfortunately, NBAR #95 field is not detected (no idea why). Furthermore, it cannot even connect to the router by SNMP. The application hangs frequently and it is quite unusable in our approach.

E. Other tools

   We did not find any other tools which should support NetFlow v9 format together with the #95 field.

Filtering of the results

The results must be filtered to remove any debug information and the headers. Additionally, we need to filter all flows which were associated directly with the router which was used for the classification by NBAR or with the local network where the router existed. There are many broadcasts and multicasts, Cisco Discovery Protocol flows, etc. So, we need to leave flows which are associated only with the original clients. We can do that using the IP addresses of the clients (they did not change during the experiment). The IP addresses were:

- Client 1: 147.83.42.206
- Client 2: 147.83.42.217
- Client 3: 147.83.42.187

The filtering process can be done by our *logAnalyzer* tool automatically, so we do not need to take any action. A fragment of the original output from NFACCTD is shown in Figure 3.6.

[5] `http://www.manageengine.com/products/netflow/`

```
CLASS,SRC_MAC,SRC_IP,DST_IP,SRC_PORT,DST_PORT,PROTOCOL,PACKETS,FLOWS,BYTES
dns,00:00:00:00:00:03,147.83.42.206,147.83.30.71,64217,53,udp,1,0,72
dns,00:00:00:00:00:03,147.83.30.71,147.83.42.206,53,64217,udp,1,0,214
unclassified,00:00:00:00:00:0e,147.83.42.206,84.88.81.41,3375,7774,tcp
    ,5,0,214
http,00:00:00:00:01:22,98.139.0.22,147.83.42.206,80,3637,tcp,6,0,1296
http,00:00:00:00:01:23,66.196.116.162,147.83.42.206,80,3638,tcp,5,0,653
secure-http,00:00:00:00:01:45,147.83.42.206,173.194.41.240,3413,443,tcp
    ,19,0,1800
secure-http,00:00:00:00:01:45,173.194.41.240,147.83.42.206,443,3413,tcp
    ,20,0,3490
netbios,00:00:00:00:a0:29,147.83.42.206,147.83.2.220,137,137,udp,3,0,288
netbios,00:00:00:00:a0:29,147.83.2.220,147.83.42.206,137,137,udp,3,0,270
ftp,00:00:00:01:eb:58,147.83.42.206,94.75.225.18,3266,21,tcp,11,0,546
ftp,00:00:00:01:eb:58,94.75.225.18,147.83.42.206,21,3266,tcp,15,0,1198
...
```

Figure 3.6: The original log generated by NFACCTD

3.4.7 UPC Machine Learning Classification Tool

The *dpi_benchmark* tool also allows to build, train, and test the C5.0-based Machine Learning Classification Tool. At first, we need to backup the previous classification build, then compile the software with the original classification library that will be copied to */opt/dpi_benchmark/build/classifier*:

```
cd /opt/dpi_benchmark
mv build build_orig
mkdir build
cd build
cmake ..
make
```

Now, it is the time to generate the training data for the C5.0 classifier. We run the *dpi_benchmark* tool in the *build* directory with the *-o* option and redirect the output to *classifier/upcnet.data*. In this step we use the training PCAP and INFO files:

```
./dpi_benchmark -f path/to/train/pcap/file -b path/to/train/info/file
   -o > classifier/upcnet.data
```

for example, in our case we write:

```
./dpi_benchmark -f /opt/pcapBuilder/packets_train.pcap -b
   /opt/pcapBuilder/packets_train.info -o > classifier/upcnet.data
```

As the result, we obtain the data used for training of the C5.0 classifier. The structure of each row in the *upcnet.data* file is as follows:

```
hex_src_ip,hex_dst_ip,protocol,src_port,dst_port,#pkts,#bytes,ToS,
   urg,ack,push,rst,syn,fin,avg_pkt_size,flow_rate,inter_time,
   flow_time,application_id
```

The description of the features and possible labels in the *upcnet.data* file can be find in a file called *upcnet.names*. Both of the files are necessary to train the C5.0 classifier. Then, we can train the classifier providing the new *upcnet.data* file:

```
cd classifier
./serviceTrainer upcnet.data 10 0 > services.data;
./c5.0 -f upcnet > class.aux;
./parse_c50.py < class.aux > tree.c
./compile.sh
cd ..
```

Now, we can process the set of the testing PCAP and INFO files by executing the following command:

```
./dpi_benchmark -f path/to/test/pcap/file -b path/to/test/info/file
   > output/file
```

for example, in our case we write:

```
./dpi_benchmark -f /opt/pcapBuilder/packets_test.pcap -b
   /opt/pcapBuilder/packets_test.info >
   /opt/pcapBuilder/mla_results_1.log
```

The format of each row in the log files is identical as in the first approach:

```
id#initial_ts#final_ts#src_ip#dst_ip#src_port#dst_port#OS#
   process_name#url#referrer#content_types#PACE_label#
   OpenDPI_label#L7_filter_label#ML_label1#ML_label2#
```

Only the MLA labels will be considered at this point, since other classifiers would be tested only on the half of the flows.

Finally, the training and testing files must be swapped and the training and classification processes must be done from the beginning. If we did not

make that, the classification by the MLA would only concern half of the flows. If we swap the training and the test sets, and re-train the classifier, we have a possibility to obtain the results for all the flows, since each flow will be once in the training set, and once in the test set.

3.5 Analysis of the classification logs

The data stored in the classification logs must be processed and imported back to the database. The most challenging part is matching the log records to the proper flows in the database. Thanks to the flow identifier contained by each flow record (either directly or encoded in the source MAC address as it is in the case of NBAR), the job can be done automatically by our *logAnalyzer* tool, which is also a part of the modified VBS system. At first, we will create a new database for the analysis purpose. This allows us to store all the data in a compact way, which is not the optimal one from the design point of view, but which speeds up the analysis process. This is thanks to many indexes (almost all columns are indexed) and due to storing the concrete string values instead of just the foreign keys. We start from creating the database for the analysis:

```
java -jar logAnalyzer.jar --createDatabase
```

In case, if for some reasons we want to drop the database and start from scratch, we can do that by:

```
java -jar logAnalyzer.jar --dropDatabase
```

The next step is to import the information about each of the flows to the new database. The information will be imported from the original INFO files, which were created during dumping the packets for analysis by *pcapBuilder*. After this step the database will contain the complete information about flows, but for now without any classification results. The job can be done by our *logAnalyzer* tool by:

```
logAnalyzer --importFlowsInformation [infoFileName] [clientId]
```

which in our case means:

```
java -jar logAnalyzer.jar --importFlowsInformation
   packets_all_1.info 1
```

```
java -jar logAnalyzer.jar --importFlowsInformation
   packets_all_2.info 2
java -jar logAnalyzer.jar --importFlowsInformation
   packets_all_3.info 3
```

Now, it is time to import the classification logs from all the tested tools.

3.5.1 PACE, OpenDPI, and the version of L7-filter used for automatic retraining purposes

The classification logs contains a lot of debug information, which can amount even for 90 % of the file size. Therefore, at first, it is good to decrease the size of the classification log by removing the unnecessary lines by fast Linux GREP tool:

```
grep "#" packets_all_1.log > packets_all_1.cleanlog
grep "#" packets_all_2.log > packets_all_2.cleanlog
grep "#" packets_all_3.log > packets_all_3.cleanlog
```

That step is not necessary, as *logAnalyzer* can handle the raw output of the classifiers, but it greatly enhances the speed. Afterwards, it is sufficient to run our *logAnalyzer* tool to import the classification results into the database:

```
java -jar logAnalyzer.jar --importDPILogs packets_all_1.cleanlog
java -jar logAnalyzer.jar --importDPILogs packets_all_2.cleanlog
java -jar logAnalyzer.jar --importDPILogs packets_all_3.cleanlog
```

3.5.2 L7-filter – the standard versions

The raw outputs of L7-filter classifications do not have any debug information, so no prior filtering of the log is advised. It is sufficient to run our *logAnalyzer* tool to import the classification results into the database. For the first version of the classifier, which has activated all the patterns:

```
java -jar logAnalyzer.jar --importL7AllLog packets_all_1.l7all
java -jar logAnalyzer.jar --importL7AllLog packets_all_2.l7all
java -jar logAnalyzer.jar --importL7AllLog packets_all_3.l7all
```

For the second version of the classifier, which does not have activated patterns declared as *overmatching*:

```
java -jar logAnalyzer.jar --importL7SelLog packets_all_1.l7sel
java -jar logAnalyzer.jar --importL7SelLog packets_all_2.l7sel
java -jar logAnalyzer.jar --importL7SelLog packets_all_3.l7sel
```

3.5.3 L7-filter – Computer Networks version

The raw outputs of L7-filter classifications do not have any debug information, so no prior filtering of the log is advised. It is sufficient to run our *logAnalyzer* tool to import the classification results into the database. For the first version of the classifier, which has activated all the patterns:

```
java -jar logAnalyzer.jar --importL7ComNetLog
   packets_all_1.l7filter_comnet
java -jar logAnalyzer.jar --importL7ComNetLog
   packets_all_2.l7filter_comnet
java -jar logAnalyzer.jar --importL7ComNetLog
   packets_all_3.l7filter_comnet
```

3.5.4 NDPI

The raw output of NDPI classification does not have any debug information, so no prior filtering of the log is advised. It is sufficient to run our *logAnalyzer* tool to import the classification results into the database:

```
java -jar logAnalyzer.jar --importNDPILog packets_all_1.ndpi_output
java -jar logAnalyzer.jar --importNDPILog packets_all_2.ndpi_output
java -jar logAnalyzer.jar --importNDPILog packets_all_3.ndpi_output
```

3.5.5 Libprotoident

The raw output of Libprotoident classification does not have any debug information, so no prior filtering of the log is advised. It is sufficient to run our *logAnalyzer* tool to import the classification results into the database:

```
java -jar logAnalyzer.jar --importLibprotoidentLog
   packets_all_1.libproto
java -jar logAnalyzer.jar --importLibprotoidentLog
```

```
   packets_all_2.libproto
java -jar logAnalyzer.jar --importLibprotoidentLog
   packets_all_3.libproto
```

3.5.6 NBAR

Importing of the NBAR logs to the database can be done by our *logAnalyzer* tool by:

```
logAnalyzer --importNBARLog [nbarFileName] [localIPAddress]
```

which in our case means:

```
java -jar logAnalyzer.jar --importNBARLog nbar_results_1.txt
   147.83.42.206
java -jar logAnalyzer.jar --importNBARLog nbar_results_2.txt
   147.83.42.217
java -jar logAnalyzer.jar --importNBARLog nbar_results_3.txt
   147.83.42.187
```

NBAR relies on NetFlow, which treats the flows in a unidirectional way. It means that we need to assess what is the type of the bi-directional flow based on 2 unidirectional flows (inbound and outbound). The final result of the classification is assessed in the following way:

a. Inbound and outbound flows are of the same class → the class is assigned to the bidirectional flow

b. Either inbound or outbound flow was classified as *unclassified* → the bidirectional flow gets the class from the second unidirectional flow, which was not classified as *unclassified*

c. Both inbound and outbound flows are of different classes, and none of them are *unclassified* → the bidirectional flow gets class from the unidirectional flow, which amounts for more Bytes

Estimating of the final classification result can be done by our *logAnalyzer* tool by:

```
java -jar logAnalyzer.jar --estimateFinalNBARClassification
```

3.5.7 UPC Machine Learning Classification Tool

Importing of the UPC Machine Learning Classification Tool logs to the database can be done by our *logAnalyzer* tool by:

```
logAnalyzer --importMLALog [classificationLog]
```

which in our case means:

```
java -jar logAnalyzer.jar --importMLALog mla_results_1.log
java -jar logAnalyzer.jar --importMLALog mla_results_2.log
```

UPC Machine Learning Classification Tool relies on NetFlow, which treats the flows in a unidirectional way. It means that we need to assess what is the type of the bi-directional flow based on 2 unidirectional flows (inbound and outbound). The final result of the classification is assessed in the following way:

a. Inbound and outbound flows are of the same class → the class is assigned to the bidirectional flow

b. Either inbound or outbound flow was classified as *UNKNOWN* → the bidirectional flow gets the class from the second unidirectional flow, which was not classified as *UNKNOWN*

c. Both inbound and outbound flows are of different classes, and none of them are *UNKNOWN* → the bidirectional flow gets class from the unidirectional flow, which amounts for more Bytes

Estimating of the final classification result can be done by our *logAnalyzer* tool by:

```
java -jar logAnalyzer.jar --estimateFinalMLAClassification
```

Chapter 4

Results

4.1 Criteria of the classification

Based on the data stored in the database, we can create some rules, which allow to test the accuracy of different classifiers. The classification criteria are different for each classifier, since each of the classifiers has a separate set of returned results. Furthermore, the classifiers have different coverage and granularity. There is almost unlimited number of ways in which the classifiers can be tested. At first, we show some criteria which can be used for the assessment of the results. The criteria are presented together with the proper parts of SQL statement, so they can be easily combined to test the particular classifiers.

4.1.1 Operating systems

- Windows 7

  `os = '7'`

- Windows XP

  `os = 'X'`

- Lubuntu (Ubuntu with LXDE - Lightweight X11 Desktop Environment)

  `os = 'L'`

4.1.2 Applications

Web clients

- Google Chrome (Windows 7, Windows XP, Linux)
  `process = 'chrome'`
- Mozilla Firefox (Windows 7, Windows XP, Linux)
  `process = 'firefox'`
- Microsoft Internet Explorer (Windows 7, Windows XP)
  `process = 'iexplore'`
- Mozilla Firefox plugins (Windows 7, Windows XP, Linux)
  ```
  process = 'plugin-container' OR process = 'plugin-contai'
     OR process = 'plugin-contain'
  ```

Peer-to-peer clients

A. BitTorrent clients

- uTorrent (Windows 7, Windows XP)
  `process = 'uTorrent'`
- Bittorrent (Windows 7, Windows XP)
  `process = 'BitTorrent'`
- Frostwire (Windows 7, Windows XP, Linux)
  `process = 'FrostWire'`
- Vuze [Azureus] (Windows 7, Windows XP, Linux)
  `process = 'Azureus'`

B. eDonkey clients

- eMule (Windows 7, Windows XP)
  `process = 'emule'`
- aMule (Linux)
  `process = 'amule'`

FTP clients

- FileZilla (Windows 7, Windows XP, Linux)

```
process = 'filezilla'
```

- SmartFTP Client (Windows 7, Windows XP)

```
process = 'SmartFTP'
```

- CuteFTP (Windows 7, Windows XP)

```
process = 'ftpte'
```

- WinSCP (Windows 7, Windows XP)

```
process = 'WinSCP'
```

Other applications

- DNS clients (Windows 7, Windows XP, Linux)

```
(process = 'svchost' OR process = 'dnsmasq')
   AND remote_port = 53
```

- NTP clients (Windows 7, Windows XP, Linux)

```
process = 'ntpd' OR (process = 'svchost' AND
   local_port = 123 AND remote_port = 123)
```

- Remote Desktop servers (Windows 7, Windows XP, Linux)

```
(process = 'svchost' OR process = 'xrdp')
   AND local_port = 3389
```

- NETBIOS clients (Windows 7, Windows XP)

```
process = 'System' AND local_port = 137 AND
   remote_port = 137
```

- SSH server (Linux)

```
process = 'sshd' OR process = 'sshd:'
```

4.1.3 Content level

- Browser HTTP traffic

```
(process = 'chrome' OR process = 'firefox' OR
   process = 'iexplore') AND content_types <> '-'
```

- HTTP traffic containing Flash content

  ```
  content_types LIKE '%video/x-flv%' OR content_types
     LIKE '%x-shockwave-flash%'
  ```

- Browser RTMP traffic

  ```
  (process = 'chrome' OR process = 'firefox' OR
     process = 'iexplore' OR process = 'plugin-container'
     OR process = 'plugin-contai' OR process = 'plugin-
     contain') AND content_types = '-' AND (remote_ip
     LIKE '199.9.%' OR remote_ip LIKE '188.125.94.%' OR
     remote_ip LIKE '188.125.95.%') AND remote_port = 1935
  ```

  Comments:

  a. RTMP traffic can be generated by a web browser or a plugin, which is responsible for playing Flash content

  b. RTMP flows do not transport any HTTP content, so they do not have any content-type field

  c. RTMP flows use port 1935 by default (if it is not blocked)

  d. We were capturing RTMP flows from Justin.tv (IPs 199.9.x.x) and Yahoo Europe (IPs 188.125.94.x and 188.125.95.x)

  e. Even if we tried to filter out all the other types of traffic, it can happen that some minor amount of HTTP traffic also falls in this category. This can happen when the flow was not collected from the beginning (e.g. because of the packet capturer crashes, system restarts, etc)

- HTTP traffic from Google

  ```
  urls LIKE '%.google.com/%'
  ```

- HTTP traffic from Facebook

  ```
  urls LIKE '%.facebook.com/%'
  ```

- HTTP traffic from YouTube

  ```
  urls LIKE '%.youtube.com/%'
  ```

- HTTP traffic from Twitter

  ```
  urls LIKE '%.twitter.com/%'
  ```

4.2 Distribution of the flows

In this section, we give an insight into the coverage and granularity levels of each of the classifiers. We grouped all the flows, which participate in the experiment, according to the class assigned by the classifiers and ordered them by the number of flows in each class. The number of the flows in the application classes is obtained from the classifiers and it does not represent the real number of flows, which should be provided. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
678381	53.75	DNS
180661	14.32	EDONKEY_SUBTYPE_PLAIN
131817	10.44	RDP
73730	5.84	UNKNOWN
60841	4.82	HTTP_SUBTYPE_UNKNOWN
54710	4.34	BITTORRENT_SUBTYPE_PLAIN
29826	2.36	SSH
27786	2.20	NTP
10336	0.82	SSL_SUBTYPE_UNKOWN
6406	0.51	NETBIOS
3102	0.25	FLASH
1805	0.14	BITTORRENT_SUBTYPE_ENCRYPTED
1194	0.09	FTP_SUBTYPE_DATA
1068	0.08	MEEBO
146	0.01	EDONKEY_SUBTYPE_ENCRYPTED
111	0.01	FTP_SUBTYPE_CONTROL
34	0.00	QUICKTIME
31	0.00	WINDOWSMEDIA
13	0.00	IRC_SUBTYPE_UNKNOWN
8	0.00	OSCAR_SUBTYPE_UNKNOWN
7	0.00	MPEG
5	0.00	BLACKBERRY
2	0.00	OGG
1	0.00	ORB_SUBTYPE_SETUP_SERVER_CONNECTION
1	0.00	SIP_SUBTYPE_UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
678381	53.75	DNS
293641	23.27	UNKNOWN
131851	10.45	RDP
61899	4.90	HTTP

29831	2.36	SSH
27786	2.20	NTP
20333	1.61	BITTORRENT
10510	0.83	SSL
3133	0.25	FLASH
2437	0.19	NETBIOS
1344	0.11	FTP
792	0.06	EDONKEY
34	0.00	QUICKTIME
32	0.00	WINDOWSMEDIA
8	0.00	OSCAR
7	0.00	MPEG
2	0.00	OGG
1	0.00	SIP

C. L7-filter – the version developed at UPC and published in Computer Networks journal

No. of flows	% of flows	Class
675229	53.50	DNS
251021	19.89	UNKNOWN
140706	11.15	SKYPE_SUBTYPE_SKYPEOUT
68988	5.47	EDONKEY_SUBTYPE_PLAIN
35990	2.85	NTP
29868	2.37	SSH
29593	2.34	BITTORRENT_SUBTYPE_PLAIN
12690	1.01	HTTP_SUBTYPE_UNKNOWN
7899	0.63	SSL_SUBTYPE_UNKOWN
2500	0.20	SKYPE_SUBTYPE_AUDIO
2426	0.19	HTTP_SUBTYPE_CACHEHIT
1113	0.09	TSP
1020	0.08	SOCKS
913	0.07	XUNLEI
777	0.06	SSL_SUBTYPE_VALIDCERTSSL
611	0.05	RTP
218	0.02	KUGOO
162	0.01	QQ
92	0.01	FTP_SUBTYPE_CONTROL
71	0.01	HTTP_SUBTYPE_VIDEO
47	0.00	HTTP_SUBTYPE_CHACHEMISS
27	0.00	AIM_SUBTYPE_UNKNOWN
23	0.00	STUN
21	0.00	NBNS
7	0.00	PPLIVE
6	0.00	ARMAGETRON
2	0.00	SOULSEEK
1	0.00	H323
1	0.00	SIP_SUBTYPE_UNKNOWN

D. L7-filter – the version developed at UPC and used for automatic re-training purposes

No. of flows	% of flows	Class
675233	53.50	DNS
493038	39.07	UNKNOWN
29868	2.37	SSH
29644	2.35	BITTORRENT_SUBTYPE_PLAIN
12837	1.02	HTTP_SUBTYPE_UNKNOWN
9373	0.74	SSL_SUBTYPE_UNKOWN
5341	0.42	FACEBOOK
2434	0.19	HTTP_SUBTYPE_CACHEHIT
1496	0.12	YOUTUBE
1025	0.08	SOCKS
787	0.06	SSL_SUBTYPE_VALIDCERTSSL
462	0.04	TWITTER
208	0.02	QQ
92	0.01	FTP_SUBTYPE_CONTROL
48	0.00	HTTP_SUBTYPE_CHACHEMISS
43	0.00	HTTP_SUBTYPE_VIDEO
30	0.00	NBNS
27	0.00	AIM_SUBTYPE_UNKNOWN
23	0.00	STUN
8	0.00	ARMAGETRON
2	0.00	SOULSEEK
1	0.00	KUGOO
1	0.00	SIP_SUBTYPE_UNKNOWN
1	0.00	H323

E. L7-filter – the standard version with activated all patterns

No. of flows	% of flows	Class
675229	53.50	DNS
198396	15.72	UNKNOWN
140490	11.13	SKYPE_SUBTYPE_SKYPEOUT
68982	5.47	EDONKEY_SUBTYPE_PLAIN
65420	5.18	FINGER
35937	2.85	NTP
29868	2.37	SSH
29593	2.34	BITTORRENT_SUBTYPE_PLAIN
7899	0.63	SSL_SUBTYPE_UNKOWN
2730	0.22	HTTP_SUBTYPE_UNKNOWN
2480	0.20	SKYPE_SUBTYPE_AUDIO
1110	0.09	TSP
1020	0.08	SOCKS
912	0.07	XUNLEI
777	0.06	SSL_SUBTYPE_VALIDCERTSSL

609	0.05	RTP
209	0.02	KUGOO
156	0.01	QQ
92	0.01	FTP_SUBTYPE_CONTROL
27	0.00	AIM_SUBTYPE_UNKNOWN
23	0.00	STUN
21	0.00	NBNS
15	0.00	WHOIS
9	0.00	HTTP_SUBTYPE_CACHEHIT
6	0.00	PPLIVE
5	0.00	ARMAGETRON
2	0.00	SOULSEEK
2	0.00	HTTP_SUBTYPE_VIDEO
1	0.00	SIP_SUBTYPE_UNKNOWN
1	0.00	H323
1	0.00	HTTP_SUBTYPE_CHACHEMISS

F. L7-filter – the standard version without activated patterns declared as *overmatching*

No. of flows	% of flows	Class
675233	53.50	DNS
497269	39.40	UNKNOWN
29868	2.37	SSH
29608	2.35	BITTORRENT_SUBTYPE_PLAIN
14770	1.17	HTTP_SUBTYPE_UNKNOWN
9373	0.74	SSL_SUBTYPE_UNKOWN
2434	0.19	HTTP_SUBTYPE_CACHEHIT
1025	0.08	SOCKS
913	0.07	XUNLEI
787	0.06	SSL_SUBTYPE_VALIDCERTSSL
222	0.02	KUGOO
208	0.02	QQ
92	0.01	FTP_SUBTYPE_CONTROL
71	0.01	HTTP_SUBTYPE_VIDEO
48	0.00	HTTP_SUBTYPE_CHACHEMISS
30	0.00	NBNS
27	0.00	AIM_SUBTYPE_UNKNOWN
23	0.00	STUN
9	0.00	PPLIVE
8	0.00	ARMAGETRON
2	0.00	SOULSEEK
1	0.00	H323
1	0.00	SIP_SUBTYPE_UNKNOWN

G. NDPI

No. of flows	% of flows	Class
677578	53.69	dns
251975	19.97	unknown
131825	10.45	rdp
47760	3.78	http
38204	3.03	bittorrent
29831	2.36	ssh
27786	2.20	ntp
15363	1.22	google
13502	1.07	skype
9565	0.76	netbios
5889	0.47	facebook
4110	0.33	ssl
2720	0.22	flash
1452	0.12	rtp
1334	0.11	ftp
792	0.06	edonkey
735	0.06	WinUpdate
459	0.04	twitter
454	0.04	iMessage_Facetime
375	0.03	youtube
130	0.01	H323
95	0.01	Viber
27	0.00	quicktime
20	0.00	sip
13	0.00	irc
8	0.00	oscar
6	0.00	mpeg
5	0.00	http_connect
5	0.00	Last.fm
2	0.00	TeamSpeak
1	0.00	netflow
1	0.00	windowsmedia

H. Libprotoident

No. of flows	% of flows	Class
678381	53.75	DNS
205337	16.27	eMule_UDP
131529	10.42	RDP
61246	4.85	HTTP
34903	2.77	No_Payload
34048	2.70	BitTorrent_UDP
29866	2.37	SSH
27786	2.20	NTP
19314	1.53	BitTorrent

12725	1.01	Unknown_UDP
11634	0.92	Unknown_TCP
10430	0.83	HTTPS
1146	0.09	FTP_Data
803	0.06	EMule
711	0.06	RTMP
669	0.05	YahooError
474	0.04	RTP
448	0.04	Flash_Player
271	0.02	HTTP_NonStandard
92	0.01	FTP_Control
83	0.01	Invalid_Bittorrent
28	0.00	Mystery_99
23	0.00	Web_Junk
20	0.00	SIP_UDP
16	0.00	Bittorrent_Extension
10	0.00	Gnutella_UDP
9	0.00	Teredo
8	0.00	ISAKMP
6	0.00	Skype
2	0.00	DNS_TCP
2	0.00	DNS_TCP
1	0.00	HTTP_443
1	0.00	Kademlia

I. NBAR

No. of flows	% of flows	Class
678322	53.75	dns
415393	32.91	unclassified
69362	5.50	http
30807	2.44	ssh
21000	1.66	rtp
19578	1.55	bittorrent
10280	0.81	secure-http
9565	0.76	netbios
2325	0.18	rtcp
1687	0.13	skype
754	0.06	ftp
679	0.05	edonkey
496	0.04	sqlserver
449	0.04	h323
161	0.01	telnet
152	0.01	cuseeme
129	0.01	novadigm
110	0.01	ntp
98	0.01	citrix
88	0.01	mgcp

83	0.01	socks
70	0.01	skinny
66	0.01	sap
59	0.00	streamwork
45	0.00	l2tp
34	0.00	notes
31	0.00	nfs
25	0.00	sqlnet
22	0.00	pop3
22	0.00	sip
21	0.00	fasttrack
20	0.00	pptp
17	0.00	secure-imap
16	0.00	snmp
13	0.00	nntp
12	0.00	pcanywhere
10	0.00	vdolive
8	0.00	rsvp
5	0.00	kerberos
3	0.00	finger
2	0.00	irc
2	0.00	secure-pop3
1	0.00	gopher

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
678803	53.79	DNS
210900	16.71	EDONKEY_SUBTYPE_PLAIN
132955	10.54	RDP
81682	6.47	HTTP_SUBTYPE_UNKNOWN
73310	5.81	BITTORRENT_SUBTYPE_PLAIN
31243	2.48	SSH
27802	2.20	NTP
11281	0.89	SSL_SUBTYPE_UNKOWN
9631	0.76	NETBIOS
1550	0.12	FLASH
1044	0.08	MEEBO
745	0.06	FTP_SUBTYPE_DATA
367	0.03	GOOGLE
269	0.02	BITTORRENT_SUBTYPE_ENCRYPTED
200	0.02	SOCKS
148	0.01	FTP_SUBTYPE_CONTROL
80	0.01	EDONKEY_SUBTYPE_ENCRYPTED
8	0.00	HTTP_SUBTYPE_CACHEHIT
3	0.00	QUICKTIME
1	0.00	NBNS

4.3 Classification of particular types of traffic

This section provides a detailed insight into the results of classifications of different types of traffic by each of the classifiers. The classifier have different granularity, so we needed to establish a mean to compare them. Therefore, we marked the classification results in the following way:

a. The classification was correct and the returned class is on the same or higher granularity level as we were intended to obtain. These results were marked by us as *green*.

b. The classification was correct and the returned class is on lower granularity level as we were intended to obtain. These results were marked by us as *blue*.

c. The classification was obviously incorrect. These results were marked by us as *red*.

d. The classification was unsuccessful and we did not obtain any application class. These results were marked by us as *black*.

In the classification we used several versions of L7-filter and the abbreviations mean:

- L7-filter-com – UPC version, modifications are described in our Computer Networks journal paper [20]
- L7-filter-aut – UPC version, which was used for our automatic retraining mechanism [10]
- L7-filter-all – the standard version with all the patterns activated, but the patterns marked as *overmatching* have low priority
- L7-filter-sel – the standard version, the patterns marked as *overmatching* are deactivated

It is worth to notice that the number of flows reported by the classifiers for the particular traffic classes here does not need to agree with the number of flows reported before in *Distribution of the flows* section. To check the accuracy of the classifiers, we define the traffic classes in our way (the rules were shown in the *Criteria of the classification* section). For the majority of the short flows (below 10 packets) we do not have the application name taken from the system sockets, because the time of the opening of the socket was to short to be able to extract the application name. Therefore, many of the short flows were not taken into account while we were doing the evaluation below.

4.3.1 Edonkey clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	94.80	0.00	0.02	5.18
OpenDPI	0.45	0.00	0.00	99.55
L7-filter-com	34.22	0.00	8.57	57.21
L7-filter-aut	0.00	0.00	0.01	99.98
L7-filter-all	34.21	0.00	13.70	52.09
L7-filter-sel	0.00	0.00	0.04	99.96
NDPI	0.45	0.00	6.72	92.83
Libprotoident	98.39	0.00	0.00	1.60
NBAR	0.38	0.00	10.81	88.81
UPC MLA	99.78	0.00	0.22	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
167255	94.72	EDONKEY_SUBTYPE_PLAIN
9153	5.18	UNKNOWN
139	0.08	EDONKEY_SUBTYPE_ENCRYPTED
37	0.02	BITTORRENT_SUBTYPE_ENCRYPTED

B. OpenDPI

No. of flows	% of flows	Class
175795	99.55	UNKNOWN
788	0.45	EDONKEY
1	0.00	HTTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
101031	57.21	UNKNOWN

60419	34.22	EDONKEY_SUBTYPE_PLAIN
7849	4.44	SKYPE_SUBTYPE_SKYPEOUT
6031	3.42	NTP
753	0.43	SKYPE_SUBTYPE_AUDIO
366	0.21	RTP
80	0.05	TSP
34	0.02	KUGOO
14	0.01	QQ
3	0.00	NBNS
2	0.00	PPLIVE
1	0.00	AIM_SUBTYPE_UNKNOWN
1	0.00	SSL_SUBTYPE_UNKOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
176554	99.98	UNKNOWN
25	0.01	QQ
3	0.00	NBNS
1	0.00	SSL_SUBTYPE_UNKOWN
1	0.00	AIM_SUBTYPE_UNKNOWN

E. L7-filter – all patterns

No. of flows	% of flows	Class
91991	52.09	UNKNOWN
60418	34.21	EDONKEY_SUBTYPE_PLAIN
9252	5.24	FINGER
7694	4.36	SKYPE_SUBTYPE_SKYPEOUT
5996	3.40	NTP
738	0.42	SKYPE_SUBTYPE_AUDIO
364	0.21	RTP
77	0.04	TSP
33	0.02	KUGOO
14	0.01	QQ
3	0.00	NBNS
2	0.00	PPLIVE
1	0.00	SSL_SUBTYPE_UNKOWN
1	0.00	AIM_SUBTYPE_UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
176518	99.96	UNKNOWN
34	0.02	KUGOO

25	0.01	QQ
3	0.00	NBNS
2	0.00	PPLIVE
1	0.00	AIM_SUBTYPE_UNKNOWN
1	0.00	SSL_SUBTYPE_UNKOWN

G. NDPI

No. of flows	% of flows	Class
163926	92.83	unknown
11371	6.44	skype
788	0.45	edonkey
494	0.28	rtp
2	0.00	H323
1	0.00	http
1	0.00	netflow
1	0.00	TeamSpeak

H. Libprotoident

No. of flows	% of flows	Class
172949	97.94	eMule_UDP
2069	1.17	Unknown_UDP
798	0.45	EMule
653	0.37	Unknown_TCP
108	0.06	No_Payload
5	0.00	Skype
1	0.00	HTTP
1	0.00	HTTP_NonStandard

I. NBAR

No. of flows	% of flows	Class
156819	88.81	unclassified
16150	9.15	rtp
1621	0.92	rtcp
1035	0.59	skype
672	0.38	edonkey
142	0.08	cuseeme
50	0.03	streamwork
13	0.01	novadigm
12	0.01	snmp
12	0.01	h323
11	0.01	pcanywhere
9	0.01	l2tp

9	0.01	nntp
8	0.00	rsvp
8	0.00	secure-imap
4	0.00	citrix
2	0.00	finger
2	0.00	http
2	0.00	mgcp
1	0.00	gopher
1	0.00	notes
1	0.00	sap

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
176110	99.73	EDONKEY_SUBTYPE_PLAIN
316	0.18	BITTORRENT_SUBTYPE_PLAIN
62	0.04	EDONKEY_SUBTYPE_ENCRYPTED
31	0.02	BITTORRENT_SUBTYPE_ENCRYPTED
25	0.01	RDP
25	0.01	DNS
11	0.01	HTTP_SUBTYPE_UNKNOWN
4	0.00	FLASH

4.3.2 BitTorrent clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	81.44	0.00	0.01	18.54
OpenDPI	27.23	0.00	0.00	72.77
L7-filter-com	42.21	0.00	7.59	50.20
L7-filter-aut	42.26	0.00	0.28	57.46
L7-filter-all	42.17	0.00	8.78	49.05
L7-filter-sel	42.23	0.00	0.48	57.27
NDPI	56.00	0.00	0.43	43.58
Libprotoident	77.24	0.00	0.06	22.71
NBAR	27.44	0.00	1.49	71.07
UPC MLA	99.53	0.00	0.47	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
49556	78.72	BITTORRENT_SUBTYPE_PLAIN
11670	18.54	UNKNOWN
1681	2.67	BITTORRENT_SUBTYPE_ENCRYPTED
27	0.04	HTTP_SUBTYPE_UNKNOWN
7	0.01	SSL_SUBTYPE_UNKOWN
7	0.01	EDONKEY_SUBTYPE_ENCRYPTED
2	0.00	DNS
1	0.00	EDONKEY_SUBTYPE_PLAIN

B. OpenDPI

No. of flows	% of flows	Class
45812	72.77	UNKNOWN
17095	27.16	BITTORRENT
35	0.06	HTTP
7	0.01	SSL
2	0.00	DNS

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
31601	50.20	UNKNOWN
26531	42.15	BITTORRENT_SUBTYPE_PLAIN
2250	3.57	SKYPE_SUBTYPE_SKYPEOUT
796	1.26	TSP
657	1.04	EDONKEY_SUBTYPE_PLAIN
594	0.94	NTP
150	0.24	SKYPE_SUBTYPE_AUDIO
134	0.21	QQ
126	0.20	KUGOO
47	0.07	RTP
33	0.05	HTTP_SUBTYPE_UNKNOWN
10	0.02	STUN
9	0.01	NBNS
8	0.01	SSL_SUBTYPE_UNKOWN
3	0.00	DNS

2	0.00	SOULSEEK

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
36172	57.46	UNKNOWN
26561	42.19	BITTORRENT_SUBTYPE_PLAIN
150	0.24	QQ
29	0.05	HTTP_SUBTYPE_UNKNOWN
12	0.02	NBNS
11	0.02	SSL_SUBTYPE_UNKOWN
10	0.02	STUN
3	0.00	DNS
2	0.00	SOULSEEK
1	0.00	KUGOO

E. L7-filter – all patterns

No. of flows	% of flows	Class
30879	49.05	UNKNOWN
26531	42.15	BITTORRENT_SUBTYPE_PLAIN
2235	3.55	SKYPE_SUBTYPE_SKYPEOUT
796	1.26	TSP
788	1.25	FINGER
653	1.04	EDONKEY_SUBTYPE_PLAIN
587	0.93	NTP
147	0.23	SKYPE_SUBTYPE_AUDIO
128	0.20	QQ
120	0.19	KUGOO
47	0.07	RTP
10	0.02	STUN
9	0.01	NBNS
8	0.01	HTTP_SUBTYPE_UNKNOWN
8	0.01	SSL_SUBTYPE_UNKOWN
3	0.00	DNS
2	0.00	SOULSEEK

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
36054	57.27	UNKNOWN
26546	42.17	BITTORRENT_SUBTYPE_PLAIN
150	0.24	QQ
129	0.20	KUGOO
34	0.05	HTTP_SUBTYPE_UNKNOWN

12	0.02	NBNS
11	0.02	SSL_SUBTYPE_UNKOWN
10	0.02	STUN
3	0.00	DNS
2	0.00	SOULSEEK

G. NDPI

No. of flows	% of flows	Class
35141	55.82	bittorrent
27432	43.58	unknown
132	0.21	skype
105	0.17	http
95	0.15	Viber
36	0.06	rtp
7	0.01	ssl
2	0.00	dns
1	0.00	google

H. Libprotoident

No. of flows	% of flows	Class
32080	50.96	BitTorrent_UDP
16342	25.96	BitTorrent
8471	13.46	Unknown_TCP
5632	8.95	No_Payload
188	0.30	Unknown_UDP
70	0.11	Invalid_Bittorrent
54	0.09	HTTP_NonStandard
52	0.08	HTTP
24	0.04	Mystery_99
16	0.03	Bittorrent_Extension
10	0.02	Gnutella_UDP
7	0.01	HTTPS
2	0.00	DNS
1	0.00	RTP
1	0.00	Skype
1	0.00	RTMP

I. NBAR

No. of flows	% of flows	Class
44742	71.07	unclassified
17161	27.26	bittorrent
511	0.81	skype

162	0.26	rtcp
155	0.25	rtp
104	0.17	http
96	0.15	h323
8	0.01	secure-http
2	0.00	edonkey
2	0.00	novadigm
2	0.00	dns
2	0.00	mgcp
1	0.00	fasttrack
1	0.00	sap
1	0.00	notes
1	0.00	citrix

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
62331	99.02	BITTORRENT_SUBTYPE_PLAIN
207	0.33	EDONKEY_SUBTYPE_PLAIN
197	0.31	BITTORRENT_SUBTYPE_ENCRYPTED
118	0.19	HTTP_SUBTYPE_UNKNOWN
51	0.08	RDP
24	0.04	FLASH
9	0.01	SSL_SUBTYPE_UNKOWN
5	0.01	EDONKEY_SUBTYPE_ENCRYPTED
5	0.01	FTP_SUBTYPE_DATA
3	0.00	DNS
1	0.00	SOCKS

4.3.3 FTP clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	95.92	0.00	0.00	4.08
OpenDPI	96.15	0.00	0.00	3.85
L7-filter-com	6.12	0.00	86.51	7.37
L7-filter-aut	6.24	0.00	85.26	8.50
L7-filter-all	6.11	0.00	93.31	0.57
L7-filter-sel	6.24	0.00	85.26	8.50
NDPI	95.69	0.00	0.45	3.85
Libprotoident	95.58	0.00	0.00	4.42
NBAR	40.59	0.00	0.00	59.41

UPC MLA	66.67	0.00	33.33	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
775	87.87	FTP_SUBTYPE_DATA
58	6.58	FTP_SUBTYPE_CONTROL
36	4.08	UNKNOWN
8	0.91	SSL_SUBTYPE_UNKOWN
5	0.57	HTTP_SUBTYPE_UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
835	94.67	FTP
34	3.85	UNKNOWN
8	0.91	SSL
5	0.57	HTTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
748	84.81	SOCKS
65	7.37	UNKNOWN
45	5.10	FTP_SUBTYPE_CONTROL
15	1.70	SKYPE_SUBTYPE_SKYPEOUT
7	0.79	SSL_SUBTYPE_UNKOWN
1	0.11	SSL_SUBTYPE_VALIDCERTSSL
1	0.11	HTTP_SUBTYPE_UNKNOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
752	85.26	SOCKS
75	8.50	UNKNOWN
45	5.10	FTP_SUBTYPE_CONTROL
7	0.79	SSL_SUBTYPE_UNKOWN

2	0.23	HTTP_SUBTYPE_UNKNOWN
1	0.11	SSL_SUBTYPE_VALIDCERTSSL

E. L7-filter – all patterns

No. of flows	% of flows	Class
748	84.81	SOCKS
59	6.69	FINGER
45	5.10	FTP_SUBTYPE_CONTROL
15	1.70	SKYPE_SUBTYPE_SKYPEOUT
7	0.79	SSL_SUBTYPE_UNKOWN
5	0.57	UNKNOWN
1	0.11	SSL_SUBTYPE_VALIDCERTSSL
1	0.11	WHOIS
1	0.11	HTTP_SUBTYPE_UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
752	85.26	SOCKS
75	8.50	UNKNOWN
45	5.10	FTP_SUBTYPE_CONTROL
7	0.79	SSL_SUBTYPE_UNKOWN
2	0.23	HTTP_SUBTYPE_UNKNOWN
1	0.11	SSL_SUBTYPE_VALIDCERTSSL

G. NDPI

No. of flows	% of flows	Class
835	94.67	ftp
34	3.85	unknown
5	0.57	ssl
4	0.45	http
4	0.45	google

H. Libprotoident

No. of flows	% of flows	Class
784	88.89	FTP_Data
45	5.10	FTP_Control
24	2.72	Unknown_TCP
15	1.70	No_Payload
8	0.91	HTTPS

6	0.68	HTTP

I. NBAR

No. of flows	% of flows	Class
524	59.41	unclassified
344	39.00	ftp
8	0.91	secure-http
6	0.68	http

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
491	55.67	FTP_SUBTYPE_DATA
184	20.86	SOCKS
99	11.22	FLASH
61	6.92	FTP_SUBTYPE_CONTROL
29	3.29	HTTP_SUBTYPE_UNKNOWN
7	0.79	SSL_SUBTYPE_UNKOWN
7	0.79	BITTORRENT_SUBTYPE_PLAIN
3	0.34	SSH
1	0.11	EDONKEY_SUBTYPE_PLAIN

4.3.4 DNS clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	99.97	0.00	0.00	0.03
OpenDPI	99.97	0.00	0.00	0.03
L7-filter-com	98.95	0.00	0.05	1.00
L7-filter-aut	98.95	0.00	0.00	1.05
L7-filter-all	98.95	0.00	0.13	0.92
L7-filter-sel	98.95	0.00	0.00	1.05
NDPI	99.88	0.00	0.09	0.03
Libprotoident	99.97	0.00	0.00	0.04
NBAR	99.97	0.00	0.02	0.02
UPC MLA	99.98	0.00	0.02	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
6598	99.97	DNS
2	0.03	UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
6598	99.97	DNS
2	0.03	UNKNOWN

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
6531	98.95	DNS
66	1.00	UNKNOWN
2	0.03	SKYPE_SUBTYPE_SKYPEOUT
1	0.02	EDONKEY_SUBTYPE_PLAIN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
6531	98.95	DNS
69	1.05	UNKNOWN

E. L7-filter – all patterns

No. of flows	% of flows	Class
6531	98.95	DNS
61	0.92	UNKNOWN
5	0.08	FINGER
2	0.03	SKYPE_SUBTYPE_SKYPEOUT
1	0.02	EDONKEY_SUBTYPE_PLAIN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
6531	98.95	DNS
69	1.05	UNKNOWN

G. NDPI

No. of flows	% of flows	Class
6592	99.88	dns
6	0.09	rtp
2	0.03	unknown

H. Libprotoident

No. of flows	% of flows	Class
6598	99.97	DNS
1	0.02	Unknown_TCP
1	0.02	No_Payload

I. NBAR

No. of flows	% of flows	Class
6598	99.97	dns
1	0.02	unclassified
1	0.02	mgcp

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
6599	99.98	DNS
1	0.02	SSH

4.3.5 NTP clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	100.00	0.00	0.00	0.00

OpenDPI	100.00	0.00	0.00	0.00
L7-filter-com	99.83	0.00	0.15	0.02
L7-filter-aut	0.00	0.00	0.01	99.99
L7-filter-all	99.83	0.00	0.15	0.02
L7-filter-sel	0.00	0.00	0.01	99.99
NDPI	100.00	0.00	0.00	0.00
Libprotoident	100.00	0.00	0.00	0.00
NBAR	0.40	0.00	0.00	99.60
UPC MLA	100.00	0.00	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
27786	100.00	NTP

B. OpenDPI

No. of flows	% of flows	Class
27786	100.00	NTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
27739	99.83	NTP
41	0.15	EDONKEY_SUBTYPE_PLAIN
6	0.02	UNKNOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
27783	99.99	UNKNOWN
3	0.01	QQ

E. L7-filter – all patterns

No. of flows	% of flows	Class
27739	99.83	NTP
41	0.15	EDONKEY_SUBTYPE_PLAIN
6	0.02	UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
27783	99.99	UNKNOWN
3	0.01	QQ

G. NDPI

No. of flows	% of flows	Class
27786	100.00	ntp

H. Libprotoident

No. of flows	% of flows	Class
27786	100	NTP

I. NBAR

No. of flows	% of flows	Class
27676	99.60	unclassified
110	0.40	ntp

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
27786	100.00	NTP

4.3.6 Remote Desktop servers

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified

PACE	99.04	0.00	0.02	0.94
OpenDPI	99.07	0.00	0.02	0.91
L7-filter-com	0.00	0.00	91.19	8.81
L7-filter-aut	0.00	0.00	0.02	99.98
L7-filter-all	0.00	0.00	91.21	8.79
L7-filter-sel	0.00	0.00	0.02	99.98
NDPI	99.05	0.00	0.08	0.87
Libprotoident	98.83	0.00	0.16	1.01
NBAR	0.00	0.00	0.66	99.34
UPC MLA	99.79	0.00	0.21	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
131664	99.04	RDP
1243	0.94	UNKNOWN
27	0.02	HTTP_SUBTYPE_UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
131698	99.07	RDP
1209	0.91	UNKNOWN
27	0.02	HTTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
121188	91.16	SKYPE_SUBTYPE_SKYPEOUT
11712	8.81	UNKNOWN
9	0.01	RTP
8	0.01	NBNS
8	0.01	TSP
6	0.00	SKYPE_SUBTYPE_AUDIO
1	0.00	KUGOO
1	0.00	QQ

1	0.00	PPLIVE

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
132912	99.98	UNKNOWN
13	0.01	NBNS
7	0.01	QQ
2	0.00	ARMAGETRON

E. L7-filter – all patterns

No. of flows	% of flows	Class
121188	91.16	SKYPE_SUBTYPE_SKYPEOUT
11682	8.79	UNKNOWN
30	0.02	FINGER
9	0.01	RTP
8	0.01	NBNS
8	0.01	TSP
6	0.00	SKYPE_SUBTYPE_AUDIO
1	0.00	QQ
1	0.00	KUGOO
1	0.00	PPLIVE

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
132908	99.98	UNKNOWN
13	0.01	NBNS
7	0.01	QQ
3	0.00	PPLIVE
2	0.00	ARMAGETRON
1	0.00	KUGOO

G. NDPI

No. of flows	% of flows	Class
131672	99.05	rdp
1153	0.87	unknown
82	0.06	H323
27	0.02	http

H. Libprotoident

No. of flows	% of flows	Class
131376	98.83	RDP
1188	0.89	No_Payload
181	0.14	RTMP
162	0.12	Unknown_TCP
27	0.02	HTTP_NonStandard

I. NBAR

No. of flows	% of flows	Class
132052	99.34	unclassified
277	0.21	h323
105	0.08	novadigm
91	0.07	citrix
69	0.05	skinny
69	0.05	mgcp
64	0.05	sap
30	0.02	l2tp
28	0.02	notes
26	0.02	nfs
25	0.02	sqlnet
24	0.02	socks
22	0.02	http
20	0.02	fasttrack
18	0.01	pptp
11	0.01	sqlserver
2	0.00	vdolive
1	0.00	pcanywhere

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
132654	99.79	RDP
238	0.18	HTTP_SUBTYPE_UNKNOWN
22	0.02	BITTORRENT_SUBTYPE_PLAIN
14	0.01	EDONKEY_SUBTYPE_PLAIN
4	0.00	FLASH
2	0.00	EDONKEY_SUBTYPE_ENCRYPTED

4.3.7 NETBIOS clients

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	66.66	0.00	0.08	33.26
OpenDPI	24.63	0.00	0.00	75.37
L7-filter-com	0.00	0.00	8.45	91.55
L7-filter-aut	0.00	0.00	0.00	100.00
L7-filter-all	0.00	0.00	8.45	91.55
L7-filter-sel	0.00	0.00	0.00	100.00
NDPI	100.00	0.00	0.00	0.00
Libprotoident	0.00	0.00	5.03	94.97
NBAR	100.00	0.00	0.00	0.00
UPC MLA	100.00	0.00	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
6296	66.66	NETBIOS
3141	33.26	UNKNOWN
8	0.08	EDONKEY_SUBTYPE_PLAIN

B. OpenDPI

No. of flows	% of flows	Class
7119	75.37	UNKNOWN
2326	24.63	NETBIOS

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
8647	91.55	UNKNOWN
407	4.31	SKYPE_SUBTYPE_SKYPEOUT
200	2.12	EDONKEY_SUBTYPE_PLAIN
98	1.04	NTP
93	0.98	RTP

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
9445	100.00	UNKNOWN

E. L7-filter – all patterns

No. of flows	% of flows	Class
8647	91.55	UNKNOWN
407	4.31	SKYPE_SUBTYPE_SKYPEOUT
200	2.12	EDONKEY_SUBTYPE_PLAIN
98	1.04	NTP
93	0.98	RTP

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
9445	100.00	UNKNOWN

G. NDPI

No. of flows	% of flows	Class
9445	100.00	netbios

H. Libprotoident

No. of flows	% of flows	Class
8970	94.97	Unknown_UDP
473	5.01	RTP
1	0.01	Kademlia
1	0.01	eMule_UDP

I. NBAR

No. of flows	% of flows	Class
9445	100.00	netbios

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class

9445	100.00	NETBIOS

4.3.8 SSH server

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	95.57	0.00	0.00	4.43
OpenDPI	95.59	0.00	0.00	4.41
L7-filter-com	95.71	0.00	0.00	4.29
L7-filter-aut	95.71	0.00	0.00	4.29
L7-filter-all	95.71	0.00	0.00	4.29
L7-filter-sel	95.71	0.00	0.00	4.29
NDPI	95.59	0.00	0.00	4.41
Libprotoident	95.71	0.00	0.00	4.30
NBAR	99.24	0.00	0.05	0.70
UPC MLA	100.00	0.00	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
25057	95.57	SSH
1162	4.43	UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
25062	95.59	SSH
1157	4.41	UNKNOWN

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class

25095	95.71	SSH
1124	4.29	UNKNOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
25095	95.71	SSH
1124	4.29	UNKNOWN

E. L7-filter – all patterns

No. of flows	% of flows	Class
25095	95.71	SSH
1124	4.29	UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
25095	95.71	SSH
1124	4.29	UNKNOWN

G. NDPI

No. of flows	% of flows	Class
25062	95.59	ssh
1157	4.41	unknown

H. Libprotoident

No. of flows	% of flows	Class
25093	95.71	SSH
1124	4.29	No_Payload
2	0.01	Unknown_TCP

I. NBAR

No. of flows	% of flows	Class
26020	99.24	ssh
184	0.70	unclassified

13	0.05	h323
1	0.00	socks
1	0.00	skinny

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
26219	100.00	SSH

4.3.9 Browser HTTP traffic

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	96.16	0.00	1.85	1.99
OpenDPI	98.01	0.00	0.00	1.99
L7-filter-com	27.02	0.00	12.19	60.79
L7-filter-aut	42.43	0.00	0.01	57.56
L7-filter-all	4.31	0.00	95.67	0.02
L7-filter-sel	31.17	0.00	1.78	67.04
NDPI	99.18	0.00	0.76	0.06
Libprotoident	98.66	0.00	0.00	1.34
NBAR	99.58	0.00	0.00	0.42
UPC MLA	98.60	0.00	1.40	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
43239	92.65	HTTP_SUBTYPE_UNKNOWN
1570	3.36	FLASH
931	1.99	UNKNOWN
863	1.85	MEEBO
31	0.07	QUICKTIME
27	0.06	WINDOWSMEDIA
6	0.01	MPEG
2	0.00	OGG

B. OpenDPI

No. of flows	% of flows	Class
44101	94.50	HTTP
1574	3.37	FLASH
927	1.99	UNKNOWN
31	0.07	QUICKTIME
27	0.06	WINDOWSMEDIA
6	0.01	MPEG
2	0.00	OGG
1	0.00	BITTORRENT

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
28371	60.79	UNKNOWN
10329	22.13	HTTP_SUBTYPE_UNKNOWN
4850	10.39	SKYPE_SUBTYPE_SKYPEOUT
2164	4.64	HTTP_SUBTYPE_CACHEHIT
826	1.77	XUNLEI
70	0.15	HTTP_SUBTYPE_VIDEO
45	0.10	HTTP_SUBTYPE_CHACHEMISS
5	0.01	ARMAGETRON
3	0.01	NTP
2	0.00	EDONKEY_SUBTYPE_PLAIN
1	0.00	SKYPE_SUBTYPE_AUDIO
1	0.00	KUGOO
1	0.00	PPLIVE
1	0.00	NBNS

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
26861	57.56	UNKNOWN
10463	22.42	HTTP_SUBTYPE_UNKNOWN
5284	11.32	FACEBOOK
2171	4.65	HTTP_SUBTYPE_CACHEHIT
1351	2.89	YOUTUBE
444	0.95	TWITTER
46	0.10	HTTP_SUBTYPE_CHACHEMISS
42	0.09	HTTP_SUBTYPE_VIDEO
5	0.01	ARMAGETRON
1	0.00	NBNS
1	0.00	SOCKS

E. L7-filter – all patterns

No. of flows	% of flows	Class
38962	83.49	FINGER
4848	10.39	SKYPE_SUBTYPE_SKYPEOUT
2004	4.29	HTTP_SUBTYPE_UNKNOWN
825	1.77	XUNLEI
9	0.02	HTTP_SUBTYPE_CACHEHIT
8	0.02	UNKNOWN
4	0.01	ARMAGETRON
3	0.01	NTP
2	0.00	HTTP_SUBTYPE_VIDEO
2	0.00	EDONKEY_SUBTYPE_PLAIN
1	0.00	HTTP_SUBTYPE_CHACHEMISS
1	0.00	NBNS

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
31287	67.04	UNKNOWN
12260	26.27	HTTP_SUBTYPE_UNKNOWN
2171	4.65	HTTP_SUBTYPE_CACHEHIT
826	1.77	XUNLEI
70	0.15	HTTP_SUBTYPE_VIDEO
46	0.10	HTTP_SUBTYPE_CHACHEMISS
5	0.01	ARMAGETRON
1	0.00	NBNS
1	0.00	KUGOO
1	0.00	PPLIVE
1	0.00	SOCKS

G. NDPI

No. of flows	% of flows	Class
33590	71.97	http
5395	11.56	google
5300	11.36	facebook
1271	2.72	flash
414	0.89	twitter
356	0.76	iMessage_Facetime
281	0.60	youtube
26	0.06	unknown
24	0.05	quicktime
5	0.01	Last.fm
5	0.01	mpeg
1	0.00	H323

1	0.00	windowsmedia

H. Libprotoident

No. of flows	% of flows	Class
45937	98.43	HTTP
330	0.71	Unknown_TCP
296	0.63	No_Payload
86	0.18	HTTP_NonStandard
18	0.04	Web_Junk
1	0.00	HTTP_443
1	0.00	FTP_Data

I. NBAR

No. of flows	% of flows	Class
46402	99.43	http
195	0.42	unclassified
70	0.15	secure-http
1	0.00	irc
1	0.00	citrix

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
45306	97.08	HTTP_SUBTYPE_UNKNOWN
644	1.38	MEEBO
487	1.04	FLASH
213	0.46	GOOGLE
12	0.03	BITTORRENT_SUBTYPE_PLAIN
3	0.01	QUICKTIME
3	0.01	HTTP_SUBTYPE_CACHEHIT
1	0.00	RDP

4.3.10 HTTP traffic containing Flash content

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	86.27	12.96	0.22	0.55
OpenDPI	86.34	13.11	0.04	0.51

L7-filter-com	1.35	13.43	12.97	72.25
L7-filter-aut	7.12	15.29	0.04	77.55
L7-filter-all	0.07	0.22	99.45	0.26
L7-filter-sel	1.35	16.97	6.17	75.50
NDPI	99.48	0.26	0.00	0.26
Libprotoident	0.00	98.07	0.00	1.93
NBAR	0.00	100.00	0.00	0.00
UPC MLA	23.85	76.01	0.15	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
2361	86.20	FLASH
355	12.96	HTTP_SUBTYPE_UNKNOWN
15	0.55	UNKNOWN
6	0.22	MEEBO
2	0.07	MPEG

B. OpenDPI

No. of flows	% of flows	Class
2363	86.27	FLASH
359	13.11	HTTP
14	0.51	UNKNOWN
2	0.07	MPEG
1	0.04	BITTORRENT

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
1979	72.25	UNKNOWN
366	13.36	HTTP_SUBTYPE_UNKNOWN
186	6.79	SKYPE_SUBTYPE_SKYPEOUT
168	6.13	XUNLEI
37	1.35	HTTP_SUBTYPE_VIDEO
2	0.07	HTTP_SUBTYPE_CACHEHIT
1	0.04	NBNS

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
2124	77.55	UNKNOWN
417	15.22	HTTP_SUBTYPE_UNKNOWN
169	6.17	YOUTUBE
17	0.62	FACEBOOK
9	0.33	HTTP_SUBTYPE_VIDEO
2	0.07	HTTP_SUBTYPE_CACHEHIT
1	0.04	NBNS

E. L7-filter – all patterns

No. of flows	% of flows	Class
2369	86.49	FINGER
186	6.79	SKYPE_SUBTYPE_SKYPEOUT
168	6.13	XUNLEI
7	0.26	UNKNOWN
6	0.22	HTTP_SUBTYPE_UNKNOWN
2	0.07	HTTP_SUBTYPE_VIDEO
1	0.04	NBNS

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
2068	75.50	UNKNOWN
463	16.90	HTTP_SUBTYPE_UNKNOWN
168	6.13	XUNLEI
37	1.35	HTTP_SUBTYPE_VIDEO
2	0.07	HTTP_SUBTYPE_CACHEHIT
1	0.04	NBNS

G. NDPI

No. of flows	% of flows	Class
2023	73.86	flash
444	16.21	google
237	8.65	youtube
17	0.62	facebook
7	0.26	http
7	0.26	unknown
3	0.11	mpeg
1	0.04	H323

H. Libprotoident

No. of flows	% of flows	Class
2683	97.96	HTTP
31	1.13	Unknown_TCP
22	0.80	No_Payload
3	0.11	Web_Junk

I. NBAR

No. of flows	% of flows	Class
2739	100.00	http

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
2082	76.01	HTTP_SUBTYPE_UNKNOWN
623	22.75	FLASH
29	1.06	GOOGLE
4	0.15	BITTORRENT_SUBTYPE_PLAIN
1	0.04	QUICKTIME

4.3.11 Browser RTMP traffic

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	0.00	80.56	0.00	19.44
OpenDPI	0.00	82.44	0.00	17.56
L7-filter-com	0.00	0.00	23.19	76.81
L7-filter-aut	0.00	0.00	0.23	99.77
L7-filter-all	0.00	0.00	24.12	75.88
L7-filter-sel	0.00	0.00	0.94	99.06
NDPI	0.00	78.92	8.90	12.18
Libprotoident	77.28	0.00	0.47	22.25
NBAR	0.00	0.23	0.23	99.53
UPC MLA	0.00	72.83	27.17	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
344	80.56	FLASH
83	19.44	UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
352	82.44	FLASH
75	17.56	UNKNOWN

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
328	76.81	UNKNOWN
54	12.65	SKYPE_SUBTYPE_SKYPEOUT
40	9.37	TSP
1	0.23	EDONKEY_SUBTYPE_PLAIN
1	0.23	KUGOO
1	0.23	H323
1	0.23	SKYPE_SUBTYPE_AUDIO
1	0.23	PPLIVE

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
426	99.77	UNKNOWN
1	0.23	H323

E. L7-filter – all patterns

No. of flows	% of flows	Class
324	75.88	UNKNOWN
54	12.65	SKYPE_SUBTYPE_SKYPEOUT
40	9.37	TSP

4	0.94	FINGER
1	0.23	EDONKEY_SUBTYPE_PLAIN
1	0.23	KUGOO
1	0.23	H323
1	0.23	SKYPE_SUBTYPE_AUDIO
1	0.23	PPLIVE

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
423	99.06	UNKNOWN
2	0.47	KUGOO
1	0.23	H323
1	0.23	PPLIVE

G. NDPI

No. of flows	% of flows	Class
337	78.92	flash
52	12.18	unknown
38	8.90	H323

H. Libprotoident

No. of flows	% of flows	Class
330	77.28	RTMP
65	15.22	No_Payload
30	7.03	Unknown_TCP
2	0.47	SSL/TLS

I. NBAR

No. of flows	% of flows	Class
425	99.53	unclassified
1	0.23	http
1	0.23	mgcp

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
311	72.83	FLASH
50	11.71	HTTP_SUBTYPE_UNKNOWN

29	6.79	RDP
24	5.62	BITTORRENT_SUBTYPE_PLAIN
9	2.11	EDONKEY_SUBTYPE_PLAIN
4	0.94	EDONKEY_SUBTYPE_ENCRYPTED

4.3.12 HTTP traffic from Google

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	0.00	100.00	0.00	0.00
OpenDPI	0.00	100.00	0.00	0.00
L7-filter-com	0.00	76.73	9.19	14.07
L7-filter-aut	0.00	82.75	0.45	16.80
L7-filter-all	0.00	0.57	99.43	0.00
L7-filter-sel	0.00	82.75	0.45	16.80
NDPI	97.28	2.61	0.11	0.00
Libprotoident	0.00	96.37	0.00	3.63
NBAR	0.00	100.00	0.00	0.00
UPC MLA	2.27	97.73	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
881	100.00	HTTP_SUBTYPE_UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
881	100.00	HTTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class

676	76.73	HTTP_SUBTYPE_UNKNOWN
124	14.07	UNKNOWN
77	8.74	SKYPE_SUBTYPE_SKYPEOUT
4	0.45	ARMAGETRON

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
729	82.75	HTTP_SUBTYPE_UNKNOWN
148	16.80	UNKNOWN
4	0.45	ARMAGETRON

E. L7-filter – all patterns

No. of flows	% of flows	Class
796	90.35	FINGER
77	8.74	SKYPE_SUBTYPE_SKYPEOUT
5	0.57	HTTP_SUBTYPE_UNKNOWN
3	0.34	ARMAGETRON

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
729	82.75	HTTP_SUBTYPE_UNKNOWN
148	16.80	UNKNOWN
4	0.45	ARMAGETRON

G. NDPI

No. of flows	% of flows	Class
857	97.28	google
23	2.61	http
1	0.11	Last.fm

H. Libprotoident

No. of flows	% of flows	Class
849	96.37	HTTP
27	3.06	Unknown_TCP
5	0.57	No_Payload

I. NBAR

No. of flows	% of flows	Class
881	100.00	http

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
861	97.73	HTTP_SUBTYPE_UNKNOWN
15	1.70	FLASH
4	0.45	GOOGLE
1	0.11	QUICKTIME

4.3.13 HTTP traffic from Facebook

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	0.00	100.00	0.00	0.00
OpenDPI	0.00	100.00	0.00	0.00
L7-filter-com	0.00	18.39	18.76	62.85
L7-filter-aut	100.00	0.00	0.00	0.00
L7-filter-all	0.00	10.78	89.22	0.00
L7-filter-sel	0.00	19.07	0.00	80.93
NDPI	100.00	0.00	0.00	0.00
Libprotoident	0.00	99.17	0.00	0.83
NBAR	0.00	99.77	0.00	0.23
UPC MLA	0.00	100.00	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
1327	100.00	HTTP_SUBTYPE_UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
1327	100.00	HTTP

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
834	62.85	UNKNOWN
249	18.76	SKYPE_SUBTYPE_SKYPEOUT
244	18.39	HTTP_SUBTYPE_UNKNOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
1327	100.00	FACEBOOK

E. L7-filter – all patterns

No. of flows	% of flows	Class
935	70.46	FINGER
249	18.76	SKYPE_SUBTYPE_SKYPEOUT
143	10.78	HTTP_SUBTYPE_UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
1074	80.93	UNKNOWN
253	19.07	HTTP_SUBTYPE_UNKNOWN

G. NDPI

No. of flows	% of flows	Class
1327	100.00	facebook

H. Libprotoident

No. of flows	% of flows	Class
1315	99.10	HTTP

6	0.45	Unknown_TCP
5	0.38	No_Payload
1	0.08	Web_Junk

I. NBAR

No. of flows	% of flows	Class
1324	99.77	http
3	0.23	unclassified

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
1327	100.00	HTTP_SUBTYPE_UNKNOWN

4.3.14 HTTP traffic from YouTube

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	34.05	64.37	0.00	1.58
OpenDPI	34.16	64.37	0.00	1.47
L7-filter-com	3.96	44.80	9.84	41.40
L7-filter-aut	49.89	1.24	0.11	48.76
L7-filter-all	0.00	0.57	98.64	0.79
L7-filter-sel	3.96	45.81	0.79	49.43
NDPI	98.65	0.45	0.00	0.90
Libprotoident	0.00	97.85	0.00	2.15
NBAR	0.00	100.00	0.00	0.00
UPC MLA	22.96	77.04	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class

569	64.37	HTTP_SUBTYPE_UNKNOWN
301	34.05	FLASH
14	1.58	UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
569	64.37	HTTP
302	34.16	FLASH
13	1.47	UNKNOWN

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
396	44.80	HTTP_SUBTYPE_UNKNOWN
366	41.40	UNKNOWN
79	8.94	SKYPE_SUBTYPE_SKYPEOUT
35	3.96	HTTP_SUBTYPE_VIDEO
6	0.68	XUNLEI
1	0.11	EDONKEY_SUBTYPE_PLAIN
1	0.11	NBNS

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
434	49.10	YOUTUBE
431	48.76	UNKNOWN
11	1.24	HTTP_SUBTYPE_UNKNOWN
7	0.79	HTTP_SUBTYPE_VIDEO
1	0.11	NBNS

E. L7-filter – all patterns

No. of flows	% of flows	Class
785	88.80	FINGER
79	8.94	SKYPE_SUBTYPE_SKYPEOUT
7	0.79	UNKNOWN
6	0.68	XUNLEI
5	0.57	HTTP_SUBTYPE_UNKNOWN
1	0.11	NBNS
1	0.11	EDONKEY_SUBTYPE_PLAIN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
437	49.43	UNKNOWN
405	45.81	HTTP_SUBTYPE_UNKNOWN
35	3.96	HTTP_SUBTYPE_VIDEO
6	0.68	XUNLEI
1	0.11	NBNS

G. NDPI

No. of flows	% of flows	Class
505	57.13	google
366	41.40	youtube
8	0.90	unknown
4	0.45	http
1	0.11	H323

H. Libprotoident

No. of flows	% of flows	Class
860	97.29	HTTP
10	1.13	Unknown_TCP
9	1.02	No_Payload
5	0.57	Web_Junk

I. NBAR

No. of flows	% of flows	Class
884	100.00	http

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
680	76.92	HTTP_SUBTYPE_UNKNOWN
167	18.89	FLASH
35	3.96	GOOGLE
1	0.11	QUICKTIME
1	0.11	HTTP_SUBTYPE_CACHEHIT

4.3.15 HTTP traffic from Twitter

Summary

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	0.00	99.25	0.00	0.75
OpenDPI	0.00	99.25	0.00	0.75
L7-filter-com	0.00	39.40	56.61	3.99
L7-filter-aut	99.50	0.50	0.00	0.00
L7-filter-all	0.00	0.50	0.00	99.50
L7-filter-sel	0.00	76.06	19.95	3.99
NDPI	99.75	0.00	0.00	0.25
Libprotoident	0.00	99.00	0.00	1.00
NBAR	0.00	100.00	0.00	0.00
UPC MLA	0.25	99.75	0.00	0.00

Results per classifier

We grouped the flows according to the class assigned by the classifiers and ordered them by the number of flows in each class. The results for the particular classifiers are shown in the following listings:

A. PACE

No. of flows	% of flows	Class
398	99.25	HTTP_SUBTYPE_UNKNOWN
3	0.75	UNKNOWN

B. OpenDPI

No. of flows	% of flows	Class
398	99.25	HTTP
3	0.75	UNKNOWN

C. L7-filter – UPC version described in Computer Networks journal

No. of flows	% of flows	Class
158	39.40	HTTP_SUBTYPE_UNKNOWN
147	36.66	SKYPE_SUBTYPE_SKYPEOUT
80	19.95	XUNLEI
16	3.99	UNKNOWN

D. L7-filter – UPC version used for automatic retraining mechanism

No. of flows	% of flows	Class
399	99.50	TWITTER
2	0.50	HTTP_SUBTYPE_UNKNOWN

E. L7-filter – all patterns

No. of flows	% of flows	Class
172	42.89	FINGER
147	36.66	SKYPE_SUBTYPE_SKYPEOUT
80	19.95	XUNLEI
2	0.50	HTTP_SUBTYPE_UNKNOWN

F. L7-filter – without *overmatching* patterns

No. of flows	% of flows	Class
305	76.06	HTTP_SUBTYPE_UNKNOWN
80	19.95	XUNLEI
16	3.99	UNKNOWN

G. NDPI

No. of flows	% of flows	Class
400	99.75	twitter
1	0.25	unknown

H. Libprotoident

No. of flows	% of flows	Class
397	99.00	HTTP
3	0.75	No_Payload
1	0.25	Unknown_TCP

I. NBAR

No. of flows	% of flows	Class
401	100.00	http

J. UPC Machine Learning Classification Tool

No. of flows	% of flows	Class
400	99.75	HTTP_SUBTYPE_UNKNOWN
1	0.25	FLASH

4.4 Summary

The following table shows the summary of classification results from all the applications / protocols together. The numbers in the table mean how many times the classifiers gave the particular result during all experiments performed by us.

Classifier	correct	correct-lg	wrong	unclassified
PACE	464451	3874	951	27453
OpenDPI	260652	3886	31	232160
L7-filter-com	159091	1842	149526	186270
L7-filter-aut	80446	1161	998	414124
L7-filter-all	148403	161	203424	144741
L7-filter-sel	72892	2157	2251	419429
NDPI	289911	371	12649	193798
Libprotoident	460438	6113	730	29448
NBAR	106950	6230	20928	362621
UPC MLA	489018	5662	2049	0

We converted the numbers to percents to better show the classification accuracy.

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	93.50	0.78	0.19	5.53
OpenDPI	52.47	0.78	0.01	46.74
L7-filter-com	32.03	0.37	30.10	37.50
L7-filter-aut	16.20	0.23	0.20	83.37
L7-filter-all	29.88	0.03	40.95	29.14
L7-filter-sel	14.67	0.43	0.45	84.44
NDPI	58.36	0.07	2.55	39.01
Libprotoident	92.69	1.23	0.15	5.93
NBAR	21.53	1.25	4.21	73.00
UPC MLA	98.45	1.14	0.41	0.00

However, in our experiment we have a big variation of number of flows originated from different application classes. Therefore, the results shown above are biased by how the particular classifiers treat application classes, which amount for the biggest number of flows in our dataset. Therefore, we decided to make second summary, where we weight each application class equally. In that case, we assign to each application class 100 units, which are distributed between *correct*, *correct-lg*, *wrong*, and *unclassified*, depending on the percentage of flows falling into the particular classifications. Then, we sum the values from all application classes and calculate the percentage. This normalized approach is not biased by the number of flows in each application class. However, it is biased by the low number of application classes, granularity level of each class, and by the definition of each class. The following table shows the percentage results.

Classifier	% correct	% correct-lg	% wrong	% unclassified
PACE	63.33	30.48	0.15	6.05
OpenDPI	50.77	30.61	0.00	18.61
L7-filter-com	27.29	12.85	23.02	36.84
L7-filter-aut	36.14	6.65	5.76	51.45
L7-filter-all	25.42	0.84	48.15	25.58
L7-filter-sel	18.64	16.04	7.73	57.59
NDPI	82.73	5.48	1.17	10.61
Libprotoident	56.11	32.71	0.38	10.81
NBAR	31.17	33.33	0.88	34.61
UPC MLA	60.91	34.89	4.20	0.00

Chapter 5

Conclusion

In this report, we presented a novel approach to test different classifiers of traffic in computer networks. There are several major contributions made to the research community:

1. We showed how to build an efficient testbed, with different operating systems, by using only virtual machines.
2. We described our methodology of generating of the network traffic in a realistic way.
3. We invented a way to collect all the traffic going through the network interfaces, group it into flows, add the process label taken from the system sockets, and send packets together with all the information to the server.
4. We showed how to create and manage a system (VBS Server), which needs to deal effectively with a large MySQL database.
5. The data collected by us are available to other researchers, so they can compare the accuracy of their classifiers on the same dataset, which was already used to compare the classifiers evaluated by us.
6. We compared the accuracy of PACE, OpenDPI, NDPI, Libprotoident, NBAR, four different variants of L7-filter, and a statistic-based tool developed at UPC. The results are presented in the report.

However, our study has still many limitations and the previous conclusions refer only to the research made on our dataset:

1. Only 10 different well-known applications were included in our study and the obtained results are directly related to these applications.

2. The dataset used in our tests was generated by us using three machines. We made an effort to create the dataset manually and in a realistic way, but the dataset is still artificial. The main purpose for generating the dataset by us was the possibility to publish it together with the whole payloads. The real backbone traffic contains numerous different applications, which were not included in our dataset, and thus, in our study.

3. Therefore, the characteristics of our dataset could impact the performance of *NBAR* and *L7-filter*, which appears to be very poor. We do not disprove the numerous previous works based on these classifiers.

4. The measured performance heavily depends on the classification granularity, for example, *NDPI* can classify the traffic on the service provider level, as Yahoo, Google, or YouTube, while the rest cannot do that. Newer versions of *PACE* also can classify the traffic based on service providers, but the version used in our experiment could not do that.

5. We performed our study using full packet payloads. However, the datasets available for the research community usually carry a small part of the payload (e.g., 96 bytes) and, thus, the best solution in this case would be *Libprotoident*, as it uses the first 4 Bytes of payload for each direction.

6. *PACE* is able to detect considerably bigger number of applications than other classifiers tested by us. That makes PACE more suitable for heterogeneous scenarios.

Taking into account all the issues, we found *PACE* as the best network traffic classification solution among all the tools studied. Among non-commercial tools we would choose *Libprotoident* or *NDPI* depending on the scenario studied.

Acknowledgments

The authors want to thank Ipoque for kindly providing access to their PACE software. This research was funded by the Spanish Ministry of Science and Innovation under contract TEC2011-27474 (NOMADS project) and by the Comissionat per a Universitats i Recerca del DIUE de la Generalitat de Catalunya (ref. 2009SGR-1140). This research was also funded by Vækst-

forum Nordjylland[1], a regional Danish development project, and Bredbånd Nord A/S[2], a regional Danish Internet Service Provider.

[1]See http://www.rn.dk/RegionalUdvikling/Vaekstforum/
[2]See http://www.bredbaandnord.dk/

Bibliography

[1] Riyad Alshammari, A. Nur Zincir-Heywood. *Machine Learning based encrypted traffic classification: identifying SSH and Skype.* Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009).

[2] Sven Ubik, Petr Žejdl. *Evaluating application-layer classification using a Machine Learning technique over different high speed networks.* 2010 Fifth International Conference on Systems and Networks Communications, IEEE 2010, pp. 387–391.

[3] Jun Li, Shunyi Zhang, Yanqing Lu, Junrong Yan. *Real-time P2P traffic identification.* IEEE GLOBECOM 2008 PROCEEDINGS.

[4] Ying Zhang, Hongbo Wang, Shiduan Cheng. *A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5.* IEEE 2010, pp. 1192–1195.

[5] Jing Cai, Zhibin Zhang, Xinbo Song. *An analysis of UDP traffic classification.* IEEE 2010, pp. 116–119.

[6] Riyad Alshammari, A. Nur Zincir-Heywood. *Unveiling Skype encrypted tunnels using GP.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1–8, July 2010.

[7] Jun Li, Shunyi Zhang, Yanqing Lu, Zailong Zhang. *Internet Traffic Classification Using Machine Learning.* In Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07), pp. 239–243, August 2007.

[8] Yongli Ma, Zongjue Qian, Guochu Shou, Yihong Hu. *Study of Information Network Traffic Identification Based on C4.5 Algorithm.* In Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08), pp. 1–5, October 2008.

[9] Wei Li, Andrew W. Moore. *A Machine Learning Approach for Efficient Traffic Classification.* In Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07), pp. 310–317, October 2007.

[10] Oriol Mula-Valls. *A practical retraining mechanism for network traffic classification in operational environments.* In Master Thesis in Computer Architecture, Networks and Systems, Universitat Politècnica de Catalunya, June 2011.

[11] CAIDA Internet Data – Passive Data Sources, 2010. [Online]. Available: `http://www.caida.org/data/passive/`

[12] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: `http://vbsi.sourceforge.net/`

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

[14] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, Jens Myrup Pedersen. *Volunteer-Based System for research on the Internet traffic.* TELFOR Journal, Vol. 4, No. 1, pp. 2–7, September 2012.

[15] Tomasz Bujlow, Tahir Riaz, 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, pp. 244–248, February 2012.

[16] Tomasz Bujlow, Tahir Riaz, Jens Myrup Pedersen. *A Method for Assessing Quality of Service in Broadband Networks.* In Proceedings of the 14th International Conference on Advanced Communication Technology, pp. 826–831, February 2012.

[17] Tomasz Bujlow, Tahir Riaz, Jens Myrup Pedersen. *Classification of HTTP traffic based on C5.0 Machine Learning Algorithm.* In Proceedings of Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012), pp. 882–887, July 2012.

[18] Jens Myrup Pedersen, Tomasz Bujlow. *Obtaining Internet Flow Statistics by Volunteer-Based System*. In Fourth International Conference on Image Processing & Communications (IP&C 2012), Image Processing & Communications Challenges 4, AISC 184, pp. 261–268, September 2012.

[19] Tomasz Bujlow, Jens Myrup Pedersen. *Obtaining Application-based and Content-based Internet Traffic Statistics*. In Proceedings of the 6th International Conference on Signal Processing and Communication Systems (ICSPCS'12), pp. 1–10, December 2012.

[20] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, Josep Solé-Pareta. *Analysis of the impact of sampling on NetFlow traffic classification*. Computer Networks, Volume 55, Issue 5, pp. 1083–1099, April 2011.