Traffic Identification Engine: An Open Platform for Traffic Classification

Walter de Donato, Antonio Pescapé, and Alberto Dainotti

Abstract

The availability of open source traffic classification systems designed for both experimental and operational use, can facilitate collaboration, convergence on standard definitions and procedures, and reliable evaluation of techniques. In this article, we describe Traffic Identification Engine (TIE), an open source tool for network traffic classification, which we started developing in 2008 to promote sharing common implementations and data in this field. We designed TIE's architecture and functionalities focusing on the evaluation, comparison, and combination of different traffic classification techniques, which can be applied to both live traffic and previously captured traffic traces. Through scientific collaborations, and thanks to the support of the open source community, this platform gradually evolved over the past five years, supporting an increasing number of functionalities, some of which we highlight in this article through sample use cases.

Traffic classification (i.e., associating traffic flows with their source applications) has attracted increasing research efforts in the last decade. The explosion of this research area started when the traditional approach of relying on transport-level protocol ports became unreliable, mainly because of the increasing variety and complexity of modern Internet traffic and application-level protocols [1, 2]. Despite the large body of literature published on traffic classification, and research efforts of many academic and industry research groups, there are few open source implementations of network traffic classifiers available. In addition, one of the main issues when novel classification approaches are presented is the inability to properly evaluate and compare them [3]. While the main obstacle to performing such tasks is the actual lack of available implementations, other difficulties derive from intrinsic differences in the types of *objects* to be classified (flows, TCP connections, etc.), in the considered traffic classes (specific applications, application categories, etc.), as well as in the metrics used to evaluate classification accuracy [1].

To address these limitations, in 2008 we started the development of an open source tool for traffic classification called Traffic Identification Engine (TIE) [4]. TIE has been designed as a community-oriented tool, to provide researchers and practitioners a platform to easily implement (and share) traffic classification techniques, and enable their comparison and combination. Indeed, starting from the first release in 2009, TIE has been widely used by the traffic classification community by both academic and commercial organizations.

In this article, we first provide a brief overview of the evolution of traffic classification and the challenges addressed in the last years in this research field. We then describe the main components and functionalities of TIE by detailing some of our design choices, also driven by such analysis of the state of the art. We finally illustrate some representative use cases of applying TIE to specific research problems:

- Comparing the accuracy of different classifiers
- Comparing their classification performance
- Investigating multi-classification and combination strategies

Traffic Classification and Related Challenges

The evolution of Internet applications has made traditional methods for classifying network traffic progressively less effective [1]. Port-based approaches can easily misclassify traffic flows, mostly because of new applications reusing port numbers registered at IANA with other applications, randomly selecting port numbers, or letting users choose a preferred port. Payload-based approaches — which inspect packets content to identify peculiar patterns — are considered more reliable, but pose privacy, technological, and economic challenges, and cannot be applied to encrypted and obfuscated traffic. In addition, the increasing use of protocol encapsulation and multi-channel applications (i.e., using different communication channels for heterogeneous services) has further hindered the ability to classify Internet traffic.

To address such limitations, several alternative methods have been proposed in the literature, sometimes applying techniques and algorithms originally developed in other research fields (signal processing, statistical models, characterization of network traffic, machine learning) to various traffic properties. Examples of properties of network traffic used for such purposes are (at flow level) duration, volume, mean packet size, and (at packet level) size and inter-packet time of the first n packets of a flow. Machine-learning techniques [5] and heuristic approaches [6] have proven particularly promising when dealing with obfuscated and encrypted traffic.

Walter de Donato and Antonio Pescapé are with the University of Napoli Federico II.

Alberto Dainotti is with CAIDA at the University of California San Diego.

0890-8044/14/$25.00 © 2014 IEEE

However, progress in this area rapidly found obstacles to assessing the state of the art and reaching consesus (in terms of methodologies, definitions, and best practices) in the research and operational community. The diversity in the terminology and definitions adopted when describing approaches and metrics [2], as well as the wide range of granularities in defining flows and traffic classes across approaches [1], made it difficult to compare different studies. For instance, different approaches assign flows to traffic classes of different granularity (e.g., identifying application categories such as *peer-to-peer* vs. specific applications such as *Kazaa*, *Bittorrent*). Agreeing on shared procedures, benchmarking metrics, flow definitions, traffic classes, as well as mapping between different classification granularities (e.g., mapping the IMAP, POP, and SMTP application protocol classes to the *mail* category class) would instead yield more rigorous results and facilitate the assessment (and thus the progress) of the state of the art in this field [1]. According to this philosophy, we designed TIE to easily compare classifiers, as we show in the first use case.

Another difficulty (mostly due to privacy concerns) is in accessing traffic traces representative of different scenarios, to be used as test data or as reference for validation [1]. Inspired by solutions proposed by the community [3], we added in TIE support for sharing traffic traces without payload along with per-flow reference labels (sometimes called *ground truth*).

Most classification approaches were not designed to work in real-world scenarios (i.e., online), for example, for live reporting or triggering of actions according to classification results is expected. Several compromises have then been proposed to find the right trade-offs among accuracy, performance, and cost: reducing the amount of traffic data analyzed (e.g., limiting the number of packets inspected for each flow [7, 8]); reducing the computational overhead (e.g., shrinking the set of features [9]); exploiting the high parallelism of new computer architectures (e.g., general-purpose graphical processing units, GPGPUs [10]). We describe how we designed TIE to support online classification and performance evaluation, two key functionalities in the second use case discussed in this article.

Different classification approaches are affected by distinct limitations but show complementarity [1]. Hence, they can be combined to achieve higher accuracy using information fusion algorithms [11], which typically require additional input, such as confusion matrices[1] or Behavior Knowledge Space (BKS) tables.[2] In 2010 we started extending TIE to support intelligent combination strategies. Our third use case demonstrates such functionality.

Despite the considerable research efforts and the success of several innovative approaches, there are still no definitive answers to the challenges discussed above. As a consequence, most of the open sourse software tools do not address them. Currently, most available traffic classification tools rely on payload inspection: ***L7-filter***[3] for a long time has been the de facto standard open source DPI classifier; however, its signatures have not been updated in recent years; ***nDPI***[4] (derived by the terminated OpenDPI project) implements more up-to-date and complex pattern matching rules, also including a decoder for SSL certificates. ***libprotoident***[5] relies on the first four bytes of payload sent in each direction, the size of the first payload-bearing packet in each direction, and the TCP or UDP port numbers.

To our knowledge, there are a few (open source) traffic classifiers implementing statistical or machine-learning techniques presented in the literature: ***Tstat*** 2.0 [12] uses a customized machine-learning technique based on a Bayesian framework with *packet size* and *inter-packet time* as classification features; ***MTClass*** [13] is a multi-threaded and modular traffic classifier based on statistical approaches and supporting online classification; ***CoMo*** [14] implements two machine-learning techniques: Naive Bayes with kernel density estimation and single-class SVM; ***Diffuse***[6] applies machine-learning techniques to perform automated QoS management for traffic flows of interactive applications; and ***NeTraMark*** [15] integrates seven machine learning and two statistical classifiers into a framework similar to TIE.

Finally, a few tools are available to extract features from traffic flows or label them: ***NetAI***[7] works on both live and stored traffic; ***Fullstats***[8] supports the extraction of up to 249 features from traffic traces; ***GTVS*** [16] assists researchers in manually inspecting and semi-automatically labeling traffic traces; ***GT*** [17] associates accurate ground truth information with traffic traces in controlled environments.

The TIE Platform

In this section, we briefly describe TIE's components and functionalities by detailing some of the design choices, focused on multi-classification, comparison of approaches, and online traffic classification. For a more detailed description please refer to [4].

Definitions and Operating Modes

Definitions — In order to compare different classification approaches, TIE proposes a unified representation of classification results. It defines IDs for application classes (*applications*) and associates them with group classes (*groups*), which include applications offering similar services. Such mapping enables the comparison of techniques working at different granularities (e.g., applications vs groups) or, for instance, the comparison of traffic classifiers which have application-level protocol classes using a coarser granularity. Moreover, several application sub-classes (*sub applications*) are associated with each application, in order to discriminate related traffic flows serving different purposes (signaling vs. data, Skype voice vs. Skype chat, etc.).

Operating Modes — TIE can be run in three operating modes, each corresponding to a different overall behavior:

- ***Offline mode***: A flow is classified only when it expires or at the end of TIE execution. This mode is useful for evaluating classification techniques when no timing constraints apply, or when a classifier requires observing flows for their entire lifetime.

[1] *A confusion matrix contains in each cell (i, j) the percentage of objects of class* i *recognized by the classifier as belonging to class* j

[2] *A BKS table lists the probability of an object belonging to each class for each possible combination of outputs from different classifiers.*

[3] *http://l7-filter.sourceforge.net*

[4] *http://www.ntop.org/products/ndpi/*

[5] *http://research.wand.net.nz/software/libprotoident.php*

[6] *http://www.caia.swin.edu.au/urp/diffuse/*

[7] *http://caia.swin.edu.au/urp/dstc/netai*

[8] *https://github.com/joshsziegler/fullstats*

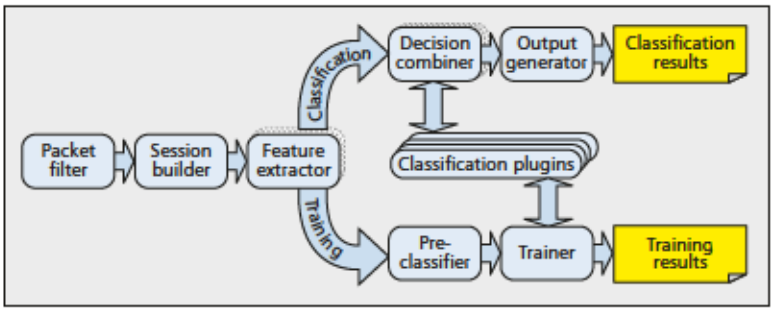


Figure 1. *Main components of the TIE engine.*

- ***Real-time mode***: A flow is classified as soon as enough information is collected, thus implementing *online* classification. This mode can be used for policy enforcement (QoS, admission control, billing, firewalling, etc.).
- ***Cyclic mode:*** Flows are classified at regular time intervals (e.g., each 5 min), and the results are stored in separate output files related only to the corresponding interval, which is useful to build live traffic reports.

All working modes can be applied to both live traffic and traffic traces. Among them, the *real-time* mode imposes most constraints and heavily influenced the whole design of the TIE engine.

Architecture Overview and Functionalities

TIE is written in C, targeting UNIX-like operating systems, currently supporting the Linux, FreeBSD, and MacOS X platforms. The software consists of a single executable and a set of classification plugins dynamically loaded at runtime. In addition, the TIE framework includes a collection of utilities — distributed with the source code — to post-process output files.

The TIE engine processes packets in five stages, with the last two varying on whether TIE is used for classifying traffic or to train machine-learning classifiers (Fig. 1).

Packet Filter — This stage captures link-layer frames — or reads them from a file — and filters them according to configurable rules. It is based on the well-known Libpcap library,[9] and its filtering capabilities are implemented using both Berkeley Packet Filters and additional user-space filtering rules (e.g., selecting traffic within a specified time range).

Session Builder — This stage organizes network traffic into sessions (i.e., the flow objects to be classified). We defined a generic concept of session to support the various types of traffic flow objects adopted in literature:

- **flow**: defined by the {SRC_{IP}, SRC_{port}, $DEST_{IP}$, $DEST_{port}$, *transport protocol*} tuple and an inactivity timeout, with a default value of 60 s.
- **biflow**: defined by the {SRC_{IP}, SRC_{port}, $DEST_{IP}$, $DEST_{port}$, *transport protocol*} tuple, where source and destination can be swapped, and the inactivity timeout is referred to packets in any direction.
- **host**: containing all the packets a host generates or receives. A timeout can be optionally set.

Except for the first session type, this stage differentiates traffic flowing in two opposite directions (*upstream* and *downstream*) by taking as reference the first observed packet (upstream). Counters, features, and state information are kept separately for each direction.

Although biflows can be considered a computationally efficient approximation of TCP connections (they only require a lookup on a hash table for each packet), some applications may need more accurate identification of their lifetime. Hence, TIE implements computationally-light heuristics based on TCP flags that, applied to biflows, yield to a better approximation of TCP connections, avoiding the segmentation of TCP connections into several biflows in presence of long periods of silence (e.g., Telnet, SSH).

This stage keeps track of sessions using a chained hash table, and — to properly work with high traffic volumes — it includes a *Garbage Collector* component responsible for periodically releasing the resources related to classified and expired sessions.

Feature Extractor — This stage is responsible for collecting the features required by the classification plugins, and is triggered by the session builder for every incoming packet. As reported in Table 1a, for each session it provides:

- Basic features (always available to classifiers)
- Advanced features (extracted on demand)

In order to optimize computational efficiency, advanced features are collected only if specified by a command line option and if a *skip-session* flag is not set (this flag avoids processing additional packets when enough packets have already been inspected). While we included support for features based on the most common classification techniques (port-based, flow-based, payload-based, etc.), TIE can easily be extended to extract new features based on definitions already published in the literature [22] or to support new techniques.

In order to rapidly experiment with techniques implemented by external tools, this stage can optionally dump for each session the corresponding classification features along with the label assigned by a classifier (e.g., a payload-based classifier can used to establish ground truth). TIE supports dumping features directly in some common formats, such as the *arff* format used by WEKA,[10] one of the most used tools in the field of machine-learning classification.

Decision Combiner — When TIE is used to classify traffic, the fourth stage of the TIE engine consists of a multi-decisional engine made of a decision combiner (hereinafter *DC*) and one or more classification plugins (hereinafter *classifiers*) implementing different classification techniques.

The DC is responsible for classifying sessions by combining multiple classifiers according to different algorithms, as reported in Table 1b. Whenever a new packet associated with an unclassified session is processed by the feature extractor, if all the classifiers are ready to be invoked on that session, the DC combines their results according to the configured algorithm in order to make the final decision. A confidence value between 0 and 100 represents the overall reliability of such a decision.

Since most combination algorithms require additional information (a sort of training of the combiner), a set of utilities extracts from a reference file (i.e., a previously generated TIE output file) the confusion matrix and BKS table necessary to train them. The default combination is the PRI, in which the classifier with higher priority determines the final result.

Classification Plugins — Traffic classification techniques are implemented in TIE as plugins exposing a standard interface [4] through which their functionalities can be activated. Each plugin is enabled only if the features it requires are available and, once enabled, its classification knowledge base is loaded.

[9] *http://www.tcpdump.org/*

[10] *http://www.cs.waikato.ac.nz/ml/weka*

(a) Supported basic and advanced per-session features

Category	Description	Availability	Packets inspected
Basic	Number of upstream/downstream packets	Always	Every packet
	Number of upstream/downstream packets carrying payload		
	Amount of upstream/downstream bytes		
	Duration		
	Source/destination port		First packet
	Transport layer protocol		
Advanced	Inter-packet time among the first *n* packets	On demand	First *n* packets First packet per direction
	Packet and payload size of the first *n* packets		
	First *n* bytes in the first packet with payload (per direction)		
	Stream of payloads up to *b* bytes		Packets necessary to collect *b* payload bytes
	Upstream/downstream payload sizes stats (min, max, mean, variance)		Every packet
	Upstream/downstream inter-packet times stats (min, max, mean, variance)		
	Full packet content (headers + payload)		
	Upstream/downstream round-trip time stats (min, max, mean, variance)	On demand (with biflow sessions only)	

(b) Implemented combination algorithms

Label	Technique	Category	Training
NB	Naive Bayes	Bayesian	Confusion matrix
MV	Majority Voting	Vote	
WMV	Weighted Majority Voting		
D-S	Dempster-Shafer	Dempster-Shafer	
BKS	BKS	Behavior knowledge space	BKS BKS & Confusion matrix
WER	Wernecke		
ORA	Oracle	Oracle	N/A
PRI	Priority-based	N/A	

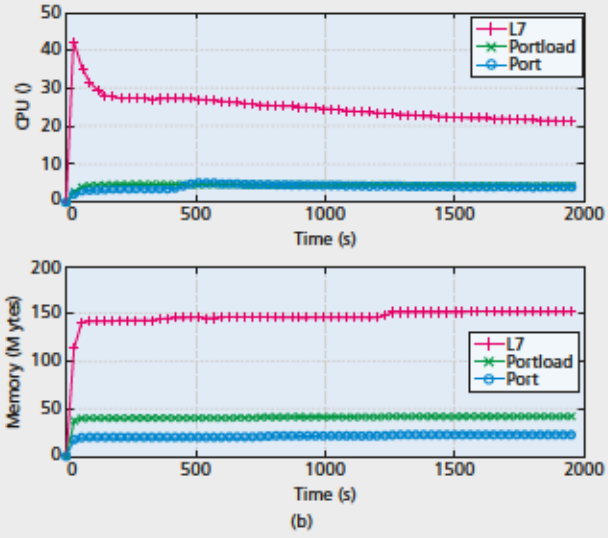
(c) Classification plugins developed since 2008

Classification plugin	Features based on	Classification approach	Collaborations and contributions from the community
Port	Protocol ports	Port-based	Developed by UNINA, signatures from CAIDA
L7	Payload	Deep payload inspection	Developed by UNINA, code and signatures from Linux L7-filter
PortLoad	Payload	Lightweight payload inspection [8]	Developed by UNINA
GMM-PS	First few packet sizes	Gaussian mixture models [7]	Developed by UNINA
HMM	Packet size and inter-packet time	Hidden Markov models [18]	Developed by UNINA
FPT	Packet size and inter-packet time	Statistical [19]	Joint work between UNINA and the University of Brescia in the context of the RECIPE research project
Joint	Packet size and inter-packet time	Nearest neighbor [20]	Joint work: UNINA, CAIDA, Seoul National University
OpenDPI	Payload	Deep payload inspection	Joint work: UNINA, TU München
WEKA/arff	Any information	Machine learning [21]	Developed by UNINA in collaboration with more than six research groups (THALES Communication and Security, Tokyo Institute of Technology, etc.)

Table 1. *Breakdown of TIE extensible functionalities.*

Classifier	Mean time (µs)	Mean time (vs. port-based)	Variance (μs^2)
Port-based	2.48	1.0	0.88
PortLoad	6.99	2.8	11.15
L7-filter	211.4	85.2	47,057.88

(a)



(b)

Figure 2. *Comparison of performance among* Port, PortLoad, *and* L7*: a) comparison of classification time (per-session average values); b) memory and CPU usage over time.*

We currently distribute TIE along with a skeleton plugin (i.e., the starting point to develop a new plugin) and two basic classification plugins, respectively implementing traditional approaches: port- and payload-based. For a more detailed description of both plugins, please refer to [4]. Since 2009, several additional plugins have been developed, also through collaborations with other research groups, implementing techniques based on machine-learning and statistical approaches (Table 1c).

Output Generator — When TIE is used to classify traffic, the last stage of the TIE engine is responsible for generating output files containing information about the sessions processed and their classification. While the output format is unique, counters and timestamps semantics depend on both the operating mode and session type [4]. When working in cyclic mode, such output can easily be processed to generate live visual reports.

Pre-Classifier — When TIE is used to train classifiers, the fourth stage of the TIE engine pre-loads the labels associated with each session from a ground-truth file, which can be obtained as output by running TIE on the same traffic trace with a ground-truth classifier.

Trainer — When TIE is used to train classifiers, the last stage of the TIE engine is responsible for invoking the signature-collection functions implemented by each enabled plugin, to let them collect the necessary per-packet information and trigger their training at the end of the TIE execution.

Extensions for Performance Evaluation — We also introduced functionalities that can be enabled at compile time and can be used — together with the native support for `gprof`[11] — to evaluate the performance of traffic classifiers. Specifically, `memory dump` and `timing` functionalities generate fine-grained memory occupation and classification time logs, respectively.

Utility Scripts — TIE is distributed along with a set of utilities for elaborating or converting output files. The most relevant is the `tie_stats` script, which rapidly produces synthetic reports and confusion matrices. A subset of scripts can instead be used to plot traffic data based on the CoralReef[12] framework, and collect and process performance related data.

TIE and the Research Community

Starting from the first release of TIE in 2009 (at that time available upon request by email), the platform has been cited in more than 40 publications and, through several collaborations, has been extended to support new classification features and schemes — including the combination of multiple classifiers — as well as to run techniques already available in WEKA (third use case). Since 2011, when a more recent version of TIE was released, TIE has been downloaded more than 150 times according to statistics collected at the official website (unique downloads of distinct users who filed a request through a web form). The download requests originated from universities (62 percent), companies (30 percent), and individuals (8 percent).

TIE at Work

This section describes the application of TIE to three different use cases. We focus on demonstrating, through practical examples, how such a tool can help tackle some of the research challenges highlighted earlier, without particular emphasis on the performance and accuracy of the classifiers used in this article.

Comparing Classification Accuracy

TIE can be used as an enabling technology for rapid development and effective comparison of traffic classification approaches in terms of accuracy. In [8], we used it to develop and evaluate a lightweight payload-based classifier called *PortLoad* (see the *PortLoad* plugin in Table 1c), which inspects only the first 32 payload bytes of the first packet in each direction of a session. We compared PortLoad to:

- A port-based classifier (the *Port* plugin, based on CoralReef signatures).
- A DPI classifier (*L7* plugin, based on L7-Filter). Such comparison has been conducted on a full-payload traffic trace of 40 Gbytes captured at the University of Napoli, Italy.

To perform this comparison, we first launched TIE with only the *L7* plugin enabled in order to use its results as a reference. We then executed TIE enabling the *PortLoad* and *Port* plugins, respectively. By running the `tie_stats` utility on the generated output files, we obtained the related confusion matrices, from which we evaluated the (expected) loss in accuracy when moving from DPI to *PortLoad* and port approaches.

As shown in Fig. 3a, *PortLoad* reported an overall accuracy and byte accuracy of about 74 and 97 percent, respectively, showing very good results on *heavy* flows. Figure 3b (right) represents the confusion matrix (with applications grouped into categories) of *PortLoad* against *L7*. The *warm* colors on the main diagonal denote a good accuracy on most (categories of) applications, whereas few cells outside of it show applica-

[11] *http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html*

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

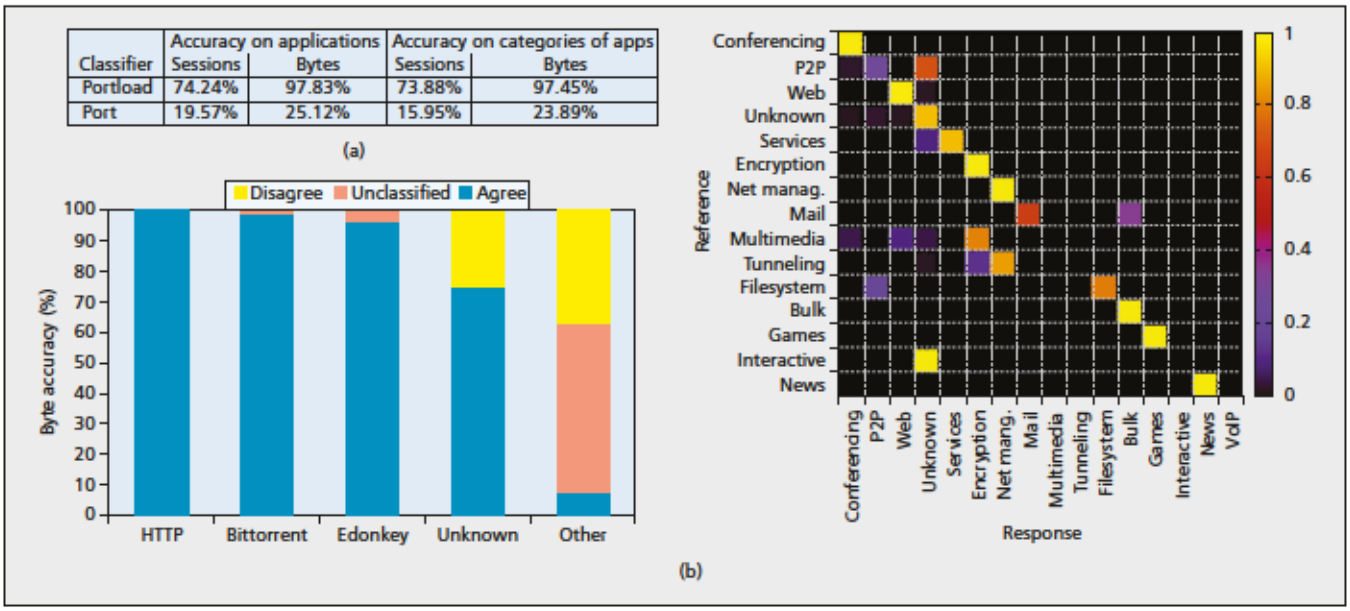


Figure 3. *Classification accuracy as obtained by comparing* Port *and* PortLoad *against L7: a) overall accuracy of* Portload *and* Port *plugins; b)* PortLoad *vs L7: byte-accuracy on applications generating most of the traffic (left); confusion matrix with applications grouped into categories (right).*

Label	Technique	Category	Features
J48	J48 Decision Tree	Machine learning	PS, IPT
K-NN	*K*-Nearest Neighbor	Machine learning	PS, IPT
R-TR	Random Tree	Machine learning	L4 protocol, biflow duration and size, PS and IPT statistics
RIP	Ripper	Machine learning	L4 protocol, biflow duration and size, PS and IPT statistics
MLP	Multi Layer Perceptron	Machine learning	PS
NBAY	Naive Bayes	Machine learning	PS
PL	Portload	Payload inspection	Payload
PORT	Port	Port	Ports

Table 2. *Classifiers used for the comparison of different combination strategies (PS: payload size, IPT: inter-packet time).*

tion categories that are not well identified by *PortLoad*. Figure 3b (left) summarizes the classification results for the applications with the largest byte counts. Each bar (corresponding to a class) represents the percentage of bytes on which *PortLoad* agreed, disagreed, or returned *unknown*, respectively, regarding *L7*. TIE allowed us to conclude that the *PortLoad* approach is a valid lightweight and privacy-preserving alternative to DPI when the main objective is to obtain high byte accuracy (97 percent) and lower unknown percentage (40 percent less).

Comparing Memory and Computational Overhead

TIE can also be used to compare the performance of traffic classification approaches in terms of classification time and CPU/memory usage. In [8], by using the same traffic trace and classifiers of the previous use case, we also compared their memory and CPU overhead using TIE's extensions for performance evaluation. We conducted our tests on GNU/Linux (kernel 2.6.27), after verifying that no other user processes were consuming significant CPU time, and none of the operating system processes were CPU- or I/O-intensive.

To measure classification time, we compiled TIE with the *timing* option enabled, and used the values measured for the *Port* plugin as a reference, since this is the fastest classification technique (it only requires one lookup on a hash table for the first packet of each session). Figure 2a summarizes the overall results with per-session average values. *PortLoad*'s average classification time was 97.5 percent lower than that of *L7*, with classification times much closer to the *Port* plugin. Such results are consistent with *PortLoad* requiring, by implementing a fixed string comparison, much less computational resources than pattern matching based on regular expression.

We monitored resource usage by enabling the *memory dump* functionality, through which heap memory allocations can be logged to file, and collecting the CPU usage of TIE with a shell script based on the *ps* command. Figure 2b shows that the CPU usage of TIE, when running only *L7*, reached the maximum value within the first 60 s after decreasing to a steady value. The slow decay is probably due to the fact that the CPU percent reported by *ps* is a moving average. On the other hand, the *Port* and *PortLoad* plugins showed similar qualitative behaviors, soon reaching steady levels.

Multi-Classification and Combination Strategies

In [21], we used TIE to perform — for the first time in the literature — a comprehensive comparison of algorithms for the combination of different traffic classification approaches. We selected different pools of classifiers out of a total of seven, the main characteristics of which are summarized in Table 2. Specifically, we considered:

- Six machine-learning-based classifiers well known in the literature, which we tested through the WEKA plugin and the support for `arff` file exporting
- The *PortLoad* payload-based classification plugin
- The *Port* plugin

(a) Classification accuracy — per-application and overall — of standalone classifiers compared to the Oracle (best values are reported in bold font).

Class	Classifier								
	J48	*K*-NN	R-TR	RIP	MLP	NBAY	PL	Port	Oracle
Bittorrent	98.8	97.4	**98.9**	98.6	55.1	79.9	7.7	21.0	99.9
SMTP	95.1	92.9	93.8	**96.0**	90.6	69.2	8.2	96.3	99.4
Skype2Skype	98.8	97.2	96.5	**99.2**	94.6	31.8	98.7	0	99.7
POP	96.0	95.0	98.7	93.9	0	79.6	29.2	**100**	100
HTTP	99.5	98.9	**99.6**	99.3	94.3	63.3	99.1	47.7	100
Soulseek	**98.6**	96.8	98.3	98.1	93	97.7	0	0	99.9
NBNS	78.4	75.9	79.9	**80.4**	9	0	0	0	85.4
QQ	0	0.7	**2.5**	0	0	0	0	0	3.2
DNS	93.6	92.6	95.3	94.4	51.1	86.2	**100**	99.7	100
SSL	96.1	93.1	95.2	93.7	69.5	68.2	**99.1**	0	98.6
RTP	**84.0**	74.1	64.5	77.3	0	41.5	0	0	92.2
EDonkey	93.0	91.7	**93.3**	91.5	72	16.1	92.9	0.1	95.7
Overall	**97.2**	95.9	96.3	97.0	82.3	43.7	83.7	15.6	98.8

(b) Classification accuracy of each combination algorithm for different pools of classifiers combined (best values and pools are reported in bold font).

Pool of classifiers	algorithm					
	NB	MV	WMV	D-S	BKS	WER
J48,R-TR,RIP	54.1	96.3	96.3	96.2	**97.7**	**97.7**
J48,R-TR,RIP,PL	55.2	96.4	96.2	96.6	**97.8**	**97.8**
J48,K-NN,R-TR,RIP,MLP	53.5	90.7	90.7	**96.7**	96.0	96.1
J48,K-NN,R-TR,RIP,MLP,NBAY	80.1	72.0	72.2	96.7	**97.3**	**97.3**
J48,K-NN,R-TR,RIP,MLP,PL	93.5	90.8	91.0	97.0	**97.9**	**97.9**
J48,K-NN,R-TR,RIP,MLP,NBAY,PL	80.9	72.0	72.2	97.0	**97.7**	**97.7**
J48,K-NN,R-TR,RIP,MLP,PL,PORT	93.6	90.5	90.8	97.1	**97.7**	**7.7**
J48,K-NN,R-TR,RIP,MLP,NBAY,PL,PORT	54.6	72.8	71.2	97.1	**97.4**	**97.4**

Table 3. *Comparison of standalone and combined classification accuracy.*

We evaluated the combination of such classifiers through the algorithms reported in Table 1b [21], implemented within the DC. The *Oracle* combiner represents the theoretic multi-classification system that correctly classifies a sample if at least one of the base classifiers is able to provide the correct classification.

We performed such analysis on a traffic trace of 59 Gbytes, counting about 1 million sessions collected at the University of Napoli. As a reference, we labeled the trace by running TIE with the *L7* plugin, filtering out all the unknown sessions (about 167,000) and all the sessions related to applications counting less than 500 occurrences. Finally, we divided (cross-validation) such a dataset in three subsets:

- 20 percent classifiers *training set*
- 40 percent classifiers and combination algorithms *validation set*
- 40 percent classifiers and combination algorithms *test set*

As a starting point, we evaluated the standalone accuracy of the classifiers, as reported in Table 3a. The different performance obtained by the classifiers for each application, and in particular their best accuracy score (printed in bold), denotes how such classifiers are complementary. Moreover, the overall accuracy obtained by the *Oracle* (98.8 percent) denotes that the combination of such classifiers theoretically could have performed better than the best classifier (97.2 percent).

We then experimented the combination of the classifiers grouping them in different pools, as shown in Table 3b, where the overall accuracies for each pool and combination algorithm are reported. The results confirmed that, in general, combination can effectively improve the classification accuracy, and such improvement depends on both the adopted combination algorithm and the selected classifiers.

(a) Accuracy obtained by each classifier when varying the number of available packets (best values are shown in bold)

Classifier	Number of packets observed for each biflow									
	1	2	3	4	5	6	7	8	9	10
J48	62.1	**94.6**	**95.9**	96.0	**96.8**	**97.1**	**97.2**	**97.2**	**97.2**	**97.2**
K-NN	62.4	91.5	92.8	95.0	94.9	94.9	95.4	95.7	95.6	95.9
R-TR	72.7	93.4	93.6	94.9	95.3	96.8	96.0	96.0	96.1	96.2
RIP	69.5	93.7	94.7	**96.2**	96.1	96.5	96.7	96.9	96.9	96.9
MLP	43.5	71.7	81.0	82.3	82.3	82.3	82.3	82.3	82.3	82.3
NBAY	31.5	39.9	42.6	43.7	43.7	43.7	43.7	43.7	43.7	43.7
PL	76.2	83.7	83.7	83.7	83.7	83.7	83.7	83.7	83.7	83.7
PORT	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6

(b) Accuracy obtained by the J48, R-TR, RIP, PL pool of classifiers when varying the number of available packets (best combination strategies are reported in bold)

Combination	Number of packets observed for each biflow									
	1	2	3	4	5	6	7	8	9	10
MV	57.8	93.9	94.4	95.6	95.9	96.2	96.3	96.3	96.4	96.4
D-S	83.1	96.0	96.9	97.0	97.4	97.4	96.4	96.5	96.5	96.5
BKS	97.0	98.4	98.3	98.4	98.4	98.4	98.4	98.4	98.4	98.4
WER	97.0	98.3	98.2	98.4	98.4	98.4	98.4	98.4	98.4	98.4

Table 4. *Classification accuracy obtained when varying the number of available packets (early classification).*

However, the *Port* and Naive Bayes classifiers — already showing low standalone performance — in general presented a negative impact on the accuracy of the multi-classifier system. The pool of classifiers achieving the best results (reported in bold) were using four and six classifiers out of the eight tested. The same table shows how the best accuracies (in bold) are achieved by the BKS and Wernecke combination algorithms, reporting the highest overall accuracy (97.9 percent).

We also investigated the performance of multi-classification systems when applied to early classification, that is, classifying traffic by relying only on the first few packets of each session. This is a realistic scenario when evaluating online traffic classification. We could easily perform such complex analysis, which was the first study of this kind in the literature [21], thanks to two key features in TIE:

- The ability to configure from which portion of traffic the features passed to the classifiers can be extracted
- Multi-classification support

We first launched TIE by separately enabling each classifier and increasing the number of packets used to extract the classification features from 1 to 10. The results, reported in Table 4a, confirmed that reducing the information available to the classifiers significantly reduced their accuracy. Such an effect was less evident for *PortLoad* and *Port* because they require at most 2 and 1 packets per session, respectively. We then applied the best four combination algorithms (according to Table 3b) to the *J48, R-TR, RIP, PL* pool of classifiers (i.e., the smallest best pool according to the previous analysis). The results, reported in Table 4b, showed an average gain in accuracy of 42 percent with respect to the standalone classifiers when extracting the features only from the first packet, which is the best condition for classifying online traffic. Hence, TIE helped us to demonstrate that multi-classification systems can dramatically improve the accuracy of online traffic classification.

Conclusion

In this article we describe TIE, a platform we started developing in 2008 to help researchers to tackle unsolved challenges in traffic classification. Thanks to the support of the open source community and scientific collaborations, the platform has gradually evolved during the past five years, enabling the production of significant scientific results. In the first quarter of 2014, we plan to release a new version of the platform based on feedback and contributions from users collected in the past two years. Thereafter, we plan to further extend TIE by:

- Investigating the optimal combination strategy and set of classifiers to generate reliable ground truth while preserving privacy
- Extending the support for sharing labeled traffic with anonymized traces
- Investigating strategies for multi-threaded classification, exploiting:
  - –Offloading techniques offered by recent traffic capturing engines such as multi-queue adapters and multi-line buses between NICs and CPU cores
  - –GPU extensions
  - –NUMA capabilities, and so on

References

[1] A. Dainotti, A. Pescapé, and K. C. Claffy, "Issues and Future Directions in Traffic Classification," *IEEE Network*, vol. 26, no. 1, 2012, pp. 35–40.

[2] A. Callado *et al.*, "A Survey on Internet Traffic Identification," *IEEE Commun. Surveys & Tutorials*, vol. 11, no. 3, 2009, pp. 37–52.

[3] L. Salgarelli, F. Gringoli, and T. Karagiannis, "Comparing Traffic Classifiers," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, 2007, pp. 65–68.
[4] A. Dainotti, W. de Donato, and A. Pescapé, "TIE: A Community-Oriented Traffic Classification Platform," *Traffic Monitoring and Analysis*, Springer, 2009, pp. 64–74.
[5] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Commun. Surveys & Tutorials*, vol. 10, no. 4, 2008, pp. 56–76.
[6] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel Traffic Classification in the Dark," *ACM SIGCOMM Comp. Commun. Review*, vol. 35, no. 4, 2005, pp. 229–40.
[7] L. Bernaille, R. Teixeira, and K. Salamatian, "Early Application Identification," *Proc. 2006 ACM CoNEXT Conf.*, 2006, p. 6.
[8] G. Aceto *et al.*, "Portload: Taking the Best of Two Worlds in Traffic Classification," *IEEE INFOCOM Wksps.*, 2010, pp. 1–5.
[9] N. Williams, S. Zander, and G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *ACM SIGCOMM CCR*, vol. 36, no. 5, Oct. 2006, pp. 7–15.
[10] G. Szabó *et al.*, "Traffic Classification over Gbit Speed with Commodity Hardware," *IEEE J. Commun. Software and Systems*, vol. 5, 2010.
[11] A. Callado *et al.*, "Better Network Traffic Identification Through the Independent Combination of Techniques," *J. Network and Computer Applications*, vol. 33, no. 4, 2010, pp. 433–46
[12] A. Finamore *et al.*, "Experiences of Internet Traffic Monitoring with tstat," *IEEE Network*, vol. 25, no. 3, 2011, pp. 8–14.
[13] F. Gringoli *et al.*, "Mtclass: Enabling Statistical Traffic Classification of Multi-Gigabit Aggregates on Inexpensive Hardware," *2012 8th Int'l. Wireless Commun. and Mobile Computing Conf.*, 2012, pp. 450–55.
[14] A. Este, F. Gringoli, and L. Salgarelli, "On-Line SVM Traffic Classification," *2011 7th Int'l. Wireless Communications and Mobile Computing Conf.*, 2011, pp. 1778–83.
[15] S. Lee *et al.*, "Netramark: A Network Traffic Classification Benchmark," *SIGCOMM Comp. Commun. Rev.*, vol. 41, no. 1, Jan. 2011, pp. 22–30.
[16] M. Canini *et al.*, "Gtvs: Boosting the Collection of Application Traffic Ground Truth," *Traffic Monitoring and Analysis*, Springer, 2009, pp. 54–63.
[17] F. Gringoli *et al.*, "Gt: Picking Up the Truth from the Ground for Internet Traffic," *Comp. Commun. Rev.*, vol. 39, no. 5, 2009, pp. 12–18.
[18] A. Dainotti *et al.*, "Classification of Network Traffic via Packet-Level Hidden Markov Models," *IEEE GLOBECOM '08*, 2008, pp. 1–5.
[19] M. Crotti *et al.*, "Traffic Classification Through Simple Statistical Fingerprinting," *ACM SIGCOMM CCR*, vol. 37, no. 1, Jan. 2007, pp. 7–16.
[20] A. Dainotti, A. Pescapé, and H. chul Kim, "Traffic Classification through Joint Distributions of Packet-Level Statistics," *IEEE GLOBECOM '11*, 2011, pp. 1–6.
[21] A. Dainotti, A. Pescapé, and C. Sansone, "Early Classification of Network Traffic through Multi-Classification," *Traffic Monitoring and Analysis*, Springer, 2011, pp. 122–35.
[22] A. Moore, D. Zuev, and M. Crogan, "Discriminators for Use in Flow-Based Classification," Dept. Comp. Sci., Queen Mary, Univ. London, tech. rep. RR-05-13, 2005.

Biographies

WALTER DE DONATO [M'06] (walter.dedonato@unina.it) is a post-doctoral researcher at the Electrical Engineering and Information Technology Department, University of Napoli Federico II, Napoli, Italy.. He received his Ph.D. degree in computer engineering and systems from the University of Napoli Federico II in 2011. His main research interests are in the field of Internet measurement and network traffic analisys, with a focus on large-scale measurement platforms. He has received several awards for his research activities.

ANTONIO PESCAPÉ [M'00, SM'09] (pescape@unina.it) is an assistant professor at the Electrical Engineering and Information Technology Department, University of Napoli Federico II. He received his Ph.D. degree in computer engineering and systems from the University of Napoli Federico II in 2004. His research interests are in the networking field with focus on Internet monitoring, measurements and management, and network security; for his research activities he has received many awards. He acts as a reviewer for national and international research and development projects.

ALBERTO DAINOTTI [M'04] (alberto@caida.org) is a research scientist at Cooperative Association for Internet Data Analysis (CAIDA), SDSC, University of California, San Diego. In 2008 he received his Ph.D. in computer engineering and systems from the Department of Computer Engineering and Systems of the University of Napoli Federico II. His main research interests are in the field of Internet measurement and network security, with a focus on the analysis of large-scale Internet events. In 2012 he was awarded the IRTF Applied Networking Research Prize.