Waterfall: Rapid Identification of IP Flows Using Cascade Classification

Paweł Foremski[1], Christian Callegari[2], and Michele Pagano[2]

[1] The Institute of Theoretical and Applied Informatics
of the Polish Academy of Sciences,
Bałtycka 5, 44-100 Gliwice, Poland
pjf@iitis.pl

[2] Department of Information Engineering, University of Pisa,
Via Caruso 16, I-56122, Italy
{c.callegari,m.pagano}@iet.unipi.it

Abstract. In the last years network traffic classification has attracted much research effort, given that it represents the foundation of many Internet functionalities such as Quality of Service (QoS) enforcement, monitoring, and security. Nonetheless, the proposed works are not able to satisfactorily solve the problem, usually being suitable for only addressing a given portion of the whole network traffic and thus none of them can be considered an ultimate solution for network classification.

In this paper, we address network traffic classification by proposing a new architecture – named Waterfall architecture – that, by combining several classification algorithms together according to a cascade principle, is able to correctly classify the whole mixture of network traffic.

Through extensive experimental tests run over real traffic datasets, we have demonstrated the effectiveness of the proposal.

Keywords: network management, traffic classification, machine learning, multi-classification, classifier selection, cascade classification.

1 Introduction

Internet traffic *classification* – or *identification* – is the act of matching IP packets to the applications that generated them [1]. For example, given the packets generated by the Skype program, a traffic classifier would group the packets in traffic *flows* and assign a *label* of *Skype* to them [2]. Traffic classification is important for network management, e.g. for Quality of Service (QoS), routing, and network diagnostics.

The field of network traffic classification needs a method for integrating results of various research activities. Many new papers describe methods that in principle propose a set of traffic features optimized for a set of network protocols [1–7]. Researchers promote their methods for classifying network traffic, which are usually quite effective, but none of them is able to exploit all observable phenomena in the Internet traffic and identify all kinds of protocols.

The question arises: could we integrate these approaches into one system, so that we move forward, building on the achievements of our colleagues? How would this improve classification systems, in terms of accuracy, functionality, completeness, and speed? Answering these questions can open new perspectives. A robust method for combining classifiers can promote research that is more focused on new phenomena in the Internet, rather than addressing the same old issues. We need a way to complement and develop our existing methods further.

This paper proposes a new, modular architecture for traffic identification systems: the Waterfall architecture. In a nutshell, it connects several classification modules in chain and query them sequentially, as long as none of them replies with a positive answer – i.e. the first module that identifies a flow wins. Typically, each module is a specialized and very accurate classifier that targets a subset of network protocols, i.e. supports the *rejection option* (the "Unknown" class) [8]. The modules are ordered from the most reliable and specific to the most general and CPU-intensive. Waterfall follows the scheme of *cascade classification*, which is a type of *classifier selection* approach in the field of *multi-classification* [9].

The proposed architecture solves the integration problem. Each module can exploit different traffic features and address different kinds of network protocols, for example traditional client-server traffic, Peer-to-Peer (P2P), or tunneled traffic. The system can be iteratively extended and updated as new network protocols emerge or new functionality requirements arise. Surprisingly, adding more classifiers can significantly reduce the total computation time (assuming proper ordering of the modules), which is the main advantage of Waterfall over popular *classifier fusion* approaches, e.g. Behavior Knowledge Space (BKS) [9, 10].

This paper presents a novel method with the following contributions:

1. It is the first application of cascade classification to the field of traffic classification (to the best of the knowledge of the authors). It represents an alternative to the BKS method (see Sect. 2 and 3).
2. Waterfall lets for integration of independent algorithms and for iterative development of traffic identification systems, in a way similar to the *divide and conquer* algorithm design paradigm (see Sect. 3 and 4).
3. It has an open source implementation in Python that shows excellent performance on real traffic and classifies flows in under 10 seconds of their lifetime (see Sect. 4 and Experiment 1 in Sect. 5).
4. Practical operation shows reduction in computation time with the increase in the number of modules, and that majority of traffic can be successfully classified using simple methods (see Experiments 1 and 2 in Sect. 5).
5. Proposes a new avenue for the future directions in the field of traffic classification (see Sect. 6, which concludes the paper).

2 Background

A naïve approach to the integration problem would be to survey recent papers for traffic features and apply them as long feature vectors classified with a decent machine learning algorithm. Even with adequate techniques employed, this could quickly lead us to the *curse of dimensionality* [8]: an exponential growth in the demand for training data as the feature space dimensionality increases. Besides, network flows differ in the set of available features, e.g. only a part of Internet flows evoke DNS queries [3]. Some features need more packets to be computed, e.g. port number is available after 1 packet, whereas payload statistics need 80 packets in [6]. This means that different tools are needed for different protocols: some flows can be classified immediately using simple methods, while others need more sophisticated analysis. Finally, from the software engineering point of view, a big, monolithic system could be hard to develop and maintain.

Instead, researchers adopt multi-classification – in particular the BKS combination method that fuses outputs of many classifiers into one final decision. In principle, the idea behind BKS is to ask all classifiers for their answers on a particular problem $\mathbf{x}$ and then query a look-up table $\mathbf{T}$ for the the final decision. The table $\mathbf{T}$ is constructed during training of the system, by observing the behavior of classifiers on a labeled dataset. For example, if an ensemble of 3 classifiers replies (A, B, A) for a sample with a ground-truth label of B, then the cell in $\mathbf{T}$ under index (A, B, A) is B (see [9], pp. 128). This powerful technique can increase the performance of traffic classification systems – as shown by Dainotti et al. in [10] – but comparing to Waterfall, it inherently requires *all* modules to be run on each traffic flow, with the drawback that the more modules are used, the more processing power is required.

In this paper, the idea of cascade classification is employed, which is also a multi-classifier, but so far it was not applied to traffic classification. Interestingly, L. Kuncheva in her book on multi-classification writes “Cascade classifiers seem to be relatively neglected although they could be of primary importance for real-life applications.” (in [9], pp. 106). This paper picks up this thought.

3 The Waterfall Architecture

The Waterfall idea is presented in Fig. 1. The input to the system is an IP flow in form of a feature vector $\mathbf{x}$, which contains all the features required by all the modules, but a particular module will usually use only a subset of $\mathbf{x}$.

The system sequentially evaluates *selection criteria* that decide which *classification modules* to use for the problem $\mathbf{x}$. If a particular criterion is fulfilled, the associated module is run. If it succeeds, the algorithm finishes. Otherwise, or if the criterion was not satisfied, the process advances to the next step. When there are no more modules to try, the flow gets rejected and is labeled as “Unknown”. More precisely,

$$Dec_i(\mathbf{x}) = \begin{cases} Class_i(\mathbf{x}) & Crit_i(\mathbf{x}) \text{ satisfied} \land Class_i(\mathbf{x}) \text{ successful} \\ Dec_{i+1}(\mathbf{x}) & \text{otherwise} \end{cases} , \tag{1}$$

$$Dec_{n+1}(\mathbf{x}) = Reject , \tag{2}$$

where Dec_i is the decision taken at step $i = \{1, 2, \ldots, n\}$, n is the number of modules, $Class_i(\mathbf{x})$ is the protocol identified by the module i, and $Crit_i(\mathbf{x})$ is the associated criterion.

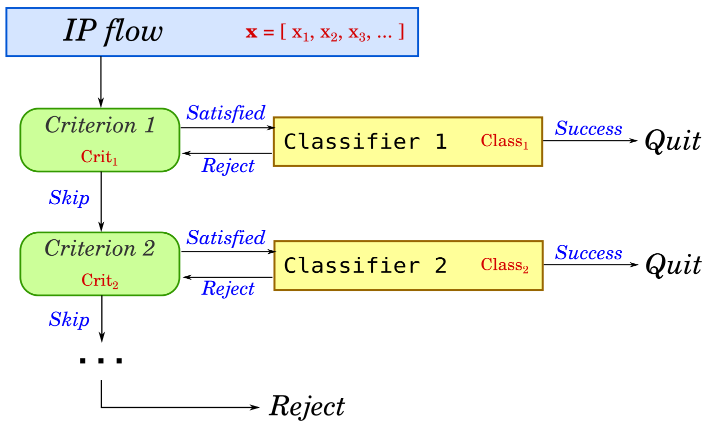

Fig. 1. The Waterfall architecture. A flow enters the system and is sequentially examined by the modules. In case of no successful classification, it is rejected.

The selection criteria are designed to skip ineligible classifiers quickly. For example, in order to implement a module that classifies traffic by analyzing the payload size of the first 5 packets in a flow, the criterion could check if at least 5 data packets were sent in each direction. If this condition is true, a machine learning algorithm could be run to identify the network protocol. Probably, a huge amount of IP flows would be skipped, saving computing resources and avoiding classification with an inadequate method. On the other hand, if a flow satisfies this criterion, it would be classified with a method that does not need to support corner cases. The selection criteria are optional, i.e. if a module does not have an associated criterion, it is always run.

4 Practical Implementation

A system using the Waterfall architecture was created and is available as open source[1]. It was implemented in the C and Python languages in two parts: *Flowcalc*, which prepares the flow feature vectors in form of ARFF files, and *Mutrics*[2], which classifies the flows. The *Mutrics* classifier has several modules, described below.

1. `dstip`: classification by destination IP address. During training, the module observes which remote destinations uniquely identify network protocols. If such particular IP address is popular enough, it is used as a rule for quick protocol identification by single lookup in a hash table.
2. `dnsclass`: classification by DNS domain name of the remote host. In [3], the authors described how to obtain the textual host names associated with

[1] See `http://mutrics.iitis.pl/`

[2] The name comes from “Multilevel Traffic Classification”.

network flows and how to use this information for traffic classification. This module implements the DNS-Class algorithm and extends it with a mechanism for detecting unknown protocols. The selection criterion checks if a particular flow has an associated DNS name or is a DNS query-response flow.

3. `portsize`: classification by the port number and packet size. In a way similar to `dstip`, the module observes which tuples of transport protocol, port number, and payload size of the first packets in both directions uniquely identify network protocols. Popular tuples are stored in a hash table. The selection criterion checks if the flow feature vector contains packet payload sizes.
4. `npkts`: classification by packet sizes. The module uses payload sizes of the first 4 packets in both directions, plus the transport protocol and the port number. It employs the random forest machine learning algorithm, which itself is a multi-classifier ensemble that combines many decision trees [9, 8]. The selection criterion is the same as for `portsize`.
5. `port`: classification by the port number. The module uses the classic pair of transport protocol and port number to find the pairs that uniquely and reliably identify network protocols, similarly to `dstip` and `portsize`. Classification requires a single lookup in a hash table.
6. `stats`: classification by flow statistics. The module uses the same machine learning algorithm as `npkts`. As features, it uses the following statistics of packet sizes and inter-arrival times in both directions: the minimum, the maximum, the average, and the standard deviation – i.e. a total of 16 statistics.

The *Flowcalc* part of the system, responsible for computing the feature vectors, was limited to only consider the first 10 seconds of traffic in each flow. This simulates a real-time scenario in which the network protocol must be identified in given time limit. All experiments presented in the next section were run with such constraints to demonstrate real-time traffic classification.

5 Experiments

In this section, the results of two experiments are presented: 1) classification performance on real network traffic, and 2) effect of adding new modules. First, the methodology is described below.

Four traffic datasets were used for experimental validation: a) *Asnet1*, collected at a Polish ISP company serving <500 residential customers, b) *Asnet2*, collected at the same network, c) *IITiS1*, collected from the network of the IITiS institute serving <50 academic users, and d) *Unibs*, collected from the campus network of the University of Brescia serving 20 workstations[3]. The *Asnet1* and *Asnet2* datasets were collected at the same gateway router, but with a time gap of 8 months. The *Asnet1* and *IITiS1* datasets were collected at different networks, but at the same time. The *Unibs* dataset was collected a few years earlier than the other datasets, contains no packet payload, and has the IP addresses anonimized. Details are presented in Table 1e.

[3] Downloaded from `http://www.ing.unibs.it/ntw/tools/traces/`

Table 1. Experiment 1: classification performance and the datasets. Metrics for validation on 4 real traffic datasets: a) *Asnet1*, b) *Asnet2*, c) *IITiS1*, and d) *Unibs*. Part e) presents details on the datasets used in the paper.

a)

Protocol	Flows	A	B	C	D	E	F	G	H	I	J	K	L	M	%TP	%FP
A = BitTorrent	799,234	99.9%								<0.1%				<0.1%	99.9%	<0.1%
B = DNS	723,478		100%												100%	0%
C = eMule	75,555			99.9%										<0.1%	99.9%	<0.1%
D = Jabber	1,388				100%										100%	0%
E = Kademlia	148,824					100%									100%	<0.1%
F = Kaspersky	17,479						100%								100%	0%
G = Mail	7,982							100%							100%	0%
H = NTP	3,506								100%						100%	0%
I = Skype	55,070	<0.1%		<0.1%		<0.1%				99.9%					99.9%	<0.1%
J = SSH	3,620										100%				100%	0%
K = Steam	50,963											100%			100%	0%
L = STUN	6,828												100%		100%	0%
M = WWW	1,695,282													100%	100%	<0.1%
	3,589,209													*Avg.*	99.9%	<0.1%

b)

Protocol	Flows	A	B	C	D	E	F	G	H	I	J	K	L	M	%TP	%FP
A = BitTorrent	4,788,510	99.9%		<0.1%						<0.1%	<0.1%		<0.1%	<0.1%	99.9%	<0.1%
B = DNS	1,810,622		100%												100%	0%
C = eMule	172,676	<0.1%		99.9%						<0.1%					99.9%	<0.1%
D = Jabber	3,110				100%										100%	0%
E = Kademlia	308,926	<0.1%				99.9%				<0.1%			<0.1%		99.9%	<0.1%
F = Kaspersky	46,475						96%							4%	96%	<0.1%
G = Mail	42,055							100%							100%	0%
H = NTP	10,940								100%						100%	0%
I = Skype	112,748	0.1%				0.5%				99.4%					99.4%	<0.1%
J = SSH	6,223										100%				100%	<0.1%
K = Steam	78,167	<0.1%										99.9%			99.9%	0%
L = STUN	12,009	<0.1%											99.9%		99.9%	<0.1%
M = WWW	3,828,118						<0.1%							99.9%	99.9%	<0.1%
	11,220,579													*Avg.*	99.6%	<0.1%

c)

Protocol	Flows	A	B	C	D	E	F	G	H	%TP	%FP
A = BitTorrent	8,811	100%								100%	0%
B = DNS	1,018,193		100%							100%	0%
C = Jabber	137			100%						100%	0%
D = Mail	108,296				100%					100%	0%
E = NTP	4,827					100%				100%	0%
F = Skype	21						100%			100%	0%
G = SSH	17,764							100%		100%	0%
H = WWW	624,511								100%	100%	0%
	1,782,560								*Avg.*	100%	0%

d)

Protocol	Flows	A	B	C	D	E	%TP	%FP
A = BitTorrent	2,928	99.8%	0.1%			<0.1%	99.8%	<0.1%
B = eMule	5,600		99.9%			<0.1%	99.9%	<0.1%
C = Mail	1,971			99.3%		0.7%	99.3%	<0.1%
D = Skype	1,484	0.1%			99.8%	0.1%	99.8%	0%
E = WWW	18,798			<0.1%		99.9%	99.9%	0.2%
	30,781					*Avg.*	99.7%	0.1%

e)

Dataset	Start	Duration	Src. IP	Dst. IP	Packets	Bytes	Avg. Util	Avg. Flows (/5 min.)	Payload
Asnet1	2012-05-26 17:40	216 h	1,828 K	1,530 K	2,525 M	1,633 G	18.0 Mbps	7.7 K	92 B
Asnet2	2013-01-24 16:26	168 h	2,503 K	2,846 K	2,766 M	1,812 G	25.7 Mbps	12.0 K	84 B
Iitis1	2012-05-26 11:19	220 h	32 K	46 K	150 M	95 G	1.0 Mbps	753.7	180 B
Unibs	2009-09-30 11:45	58 h	27	1 K	33 M	26 G	0.9 Mbps	111.7	0 B

For establishing the ground-truth labels on the datasets a)–c), Deep Packet Inspection (DPI) was employed. DPI is not perfect – as shown by Dusi et al. [11] – but it is the most popular method used in the literature, and often the only practically available. The *libprotoident* v. 2.0.7 was used as the DPI software (reported to offer very good accuracy in [12]). The *Unibs* dataset already contained ground-truth and was not suitable for DPI because of no payload data.

Finally, the datasets were sanitized by dropping flows that had no data transmitted in both directions, e.g. incomplete TCP sessions and empty UDP flows. The datasets contain different subsets of network protocols (see Table 1).

For measuring the classification accuracy for a given protocol p, the popular $\%TP_p$ and $\%FP_p$ metrics were employed:

$$\%TP_p = \frac{|TP_p|}{|F_p|} \cdot 100\,\% \ , \quad \%FP_p = \frac{|FP_p|}{|F'_p|} \cdot 100\,\% \ , \tag{3}$$

where TP_p is the set of true positives for protocol p, F_p is the set of all testing flows for protocol p that were classified, FP_p is the set of false positives for p, and F'_p is the set of all testing flows for all protocols except p that were classified. For evaluating the overall accuracy, the average values of these metrics were used – the $\%TP$ and $\%FP$ metrics – which were complemented with the $\%Unk$ metric that measures the amount of rejected flows:

$$\%Unk = \frac{|U|}{|F|} \cdot 100\,\% \ , \tag{4}$$

where U is the set of rejected flows, and F is the set of all testing flows.

For dividing the data into training and testing parts, a 60%/40% split was used on *Asnet1* and on *Unibs*, i.e. 60% of their flows were randomly selected for training, and 40% for testing. The classifier trained on *Asnet1* was validated on the rest of *Asnet1* (to evaluate the "classical" classification performance), and on the whole *Asnet2* and *IITiS1* datasets (so as to demonstrate stability in time and space). The classifier trained on *Unibs* was validated only on the rest of the *Unibs* dataset: this tested operation on a trace without packet payloads.

In the Experiment 1, the system was evaluated for classification performance on the datasets a)-d), with 5 modules enabled: `dstip`, `dnsclass`, `portsize`, `npkts`, and `port`. For the *Unibs* dataset, `stats` was used instead of `dnsclass`, because this dataset had no DNS payload packets.

The results in form of confusion matrices and performance metrics are presented in Table 1, parts a)–d). For all datasets, the $\%TP$ and $\%FP$ metrics were close to 100 % and 0 % respectively, which indicates high classification performance of the system. The classifier successfully identified all protocols, including: *BitTorrent*, *Skype*, *Kademlia*, *SSH*, *STUN*, *WWW*, and more. For the *IITiS1* dataset, the system made no errors in classifying over 1.5 million flows; for other datasets, the number of errors was well below 0.1 %. The $\%Unk$ metric was 0.1 %, 0.4 %, 1.1 %, and 1.4 %, for *Asnet1*, *Asnet2*, *IITiS1*, and *Unibs*, respectively – i.e. almost all flows were classified.

Figure 2 shows traffic progress through the system: the figure presents the percentage of IP flows at the input of successive modules. An IP flow leaves the system as soon as it gets classified, so the figure visualizes how many flows get through the end of the waterfall. It is apparent that the amount of traffic that a particular module can classify depends on the dataset. For all datasets, more than half of the flows were classified using simple methods – namely the `dstip`, `dnsclass`, and `portsize` modules – without the need to run the `npkts` module, which employs a sophisticated machine learning algorithm.

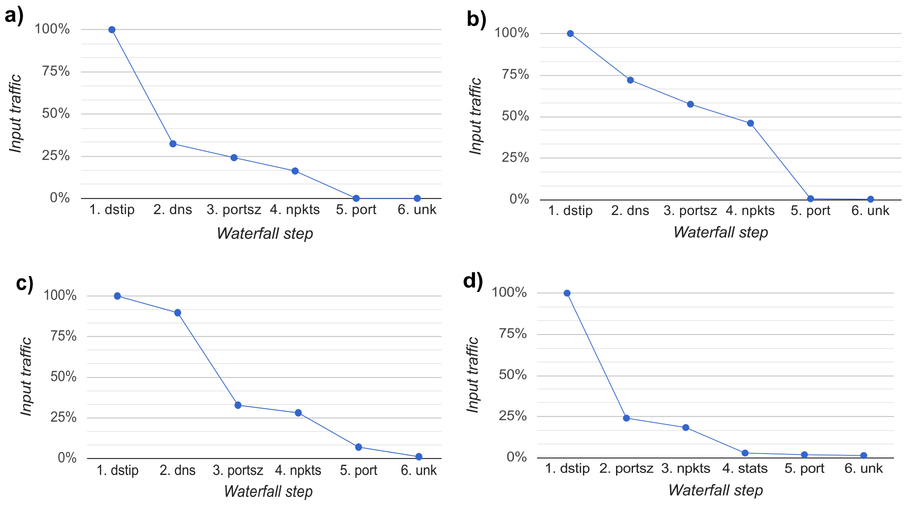

Fig. 2. Experiment 1: amount of traffic passing through successive waterfall steps, for datasets: a) *Asnet1*, b) *Asnet2*, c) *IITiS1*, and d) *Unibs*. The classifier works with 5 modules enabled; "dns" means `dnsclass` and "portsz" means `portsize`.

In the Experiment 2, the effect of increasing the number of modules was studied. The system started in configuration with only one module present: the `npkts` module, which is CPU intensive. In each iteration, one new module was added – usually at the front of the waterfall – and the whole system was given the task

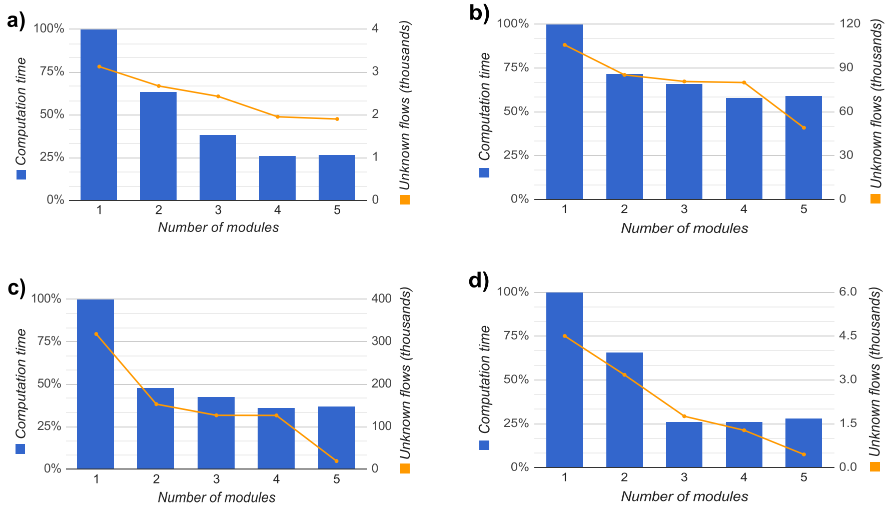

Fig. 3. Experiment 2: effect of adding modules on computation time (bars) and on the number of unknown flows (lines): a) *Asnet1*, b) *Asnet2*, c) *IITiS1*, and d) *Unibs*

of classifying the same dataset. The experiments were run in separation of each other, on a single core of an Intel Core i7 machine[4] (i.e. single-threaded implementation). The time needed to finish the computations was measured relatively to the first run. The amount of unclassified flows was measured in absolute numbers.

The results are displayed in Fig. 3. The computation time generally decreased with new modules being enabled. The modules were added in the following order: `npkts`, `dnsclass`, `portsize`, `dstip`, and `port` – that is, from most to least CPU demanding. Motivation for this was to resemble a scenario in which a generic classification algorithm is iteratively augmented with specialized methods. In each iteration, the number of unclassified flows dropped. The $\%TP$ and $\%FP$ metrics were stable and close to perfect values.

6 Conclusions

This paper presented Waterfall – a novel, modular architecture for traffic classification that lets for integration of many algorithms and exhibits a decrease in the total computation time with new modules being added. A practical, open source implementation of the system – the *Mutrics* classifier – showed very good performance results on 4 real traffic datasets, in a scenario that resembles real-time traffic classification. The paper experimentally proved that the majority of IP flows can be immediately classified using simple methods, which exploit traffic features like destination IP address, DNS domain name, and packet size.

The paper concludes with a positive answer to the question whether many independent traffic classification algorithms could be integrated, giving good results in terms of many metrics. The Waterfall architecture – together with the open source *Mutrics* classifier – are a basis for future developments.

The paper presented an alternative to the classifier fusion approach, recently introduced in the field of traffic classification. The main conclusion is that cascade classification is less computationally demanding than e.g. the BKS method. The future research could focus on a more detailed comparison between classifier fusion and classifier selection in context of traffic classification. Another interesting problems are the optimal selection of training instances for the classification modules (in accordance with their criteria), and the proper sequence of modules in the cascade.

Acknowledgments. This work was funded by the Polish National Science Centre, under research grant nr 2011/01/N/ST6/07202: project "Multilevel Traffic Classification", `http://mutrics.iitis.pl/`. This paper closes the project. The *Asnet1* and *Asnet2* traffic traces were collected thanks to ASN Sp. z o.o., `http://www.asn.pl/`.

[4] Intel Core i7-930 2.80 GHz, 8 GB RAM, 128 GB SSD.

References

1. Foremski, P.: On different ways to classify Internet traffic: a short review of selected publications. Theoretical and Applied Informatics 25(2) (2013)
2. Adami, D., Callegari, C., Giordano, S., Pagano, M., Pepe, T.: Skype-Hunter: A real-time system for the detection and classification of Skype traffic. International Journal of Communication Systems 25(3), 386–403 (2012)
3. Foremski, P., Callegari, C., Pagano, M.: DNS-Class: Immediate classification of IP flows using DNS. International Journal of Network Management (accepted, 2014)
4. Fiadino, P., Bär, A., Casas, P.: HTTPTag: A Flexible On-line HTTP Classification System for Operational 3G Networks. In: International Conference on Computer Communications, INFOCOM 2013. IEEE (2013)
5. Bermolen, P., Mellia, M., Meo, M., Rossi, D., Valenti, S.: Abacus: Accurate behavioral classification of P2P-TV traffic. Computer Networks 55(6), 1394–1411 (2011)
6. Finamore, A., Mellia, M., Meo, M., Rossi, D.: KISS: Stochastic packet inspection classifier for udp traffic. IEEE/ACM Transactions on Networking 18(5), 1505–1515 (2010)
7. Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. Computer Networks 53(1), 81–97 (2009)
8. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern classification. John Wiley & Sons (2012)
9. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley (2004)
10. Dainotti, A., Pescapé, A., Sansone, C.: Early classification of network traffic through multi-classification. Traffic Monitoring and Analysis, 122–135 (2011)
11. Dusi, M., Gringoli, F., Salgarelli, L.: Quantifying the accuracy of the ground truth associated with Internet traffic traces. Computer Networks 55(5), 1158–1167 (2011)
12. Bujlow, T., Carela-Espanol, V.: Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Polytechnic University of Catalonia (2013)