Noname manuscript No.
(will be inserted by the editor)

Waterfall traffic classification: a quick approach to optimizing cascade classifiers

Paweł Foremski · Christian Callegari · Michele Pagano

the date of receipt and acceptance should be inserted later

Abstract Heterogeneous wireless communication networks, like 4G LTE, transport diverse kinds of IP traffic: voice, video, Internet data, and more. In order to effectively manage such networks, administrators need adequate tools, of which traffic classification is the basis for visualizing, shaping, and filtering the broad streams of IP packets observed nowadays.

In this paper, we describe a modular, cascading traffic classification system—the Waterfall architecture—and we extensively describe a novel technique for its optimization—in terms of CPU time, number of errors, and percentage of unrecognized flows. We show how to significantly accelerate the process of exhaustive search for the best performing cascade. We employ 5 datasets of real Internet transmissions and 7 traffic analysis methods to demonstrate that our proposal yields valid results and outperforms a greedy optimizer.

Keywords Network Management, Convergent Networks, Traffic Classification, Machine Learning

1 Introduction

Internet traffic *classification*—or *identification*—is the act of matching IP packets with the computer program or communication protocol that generated them [1]. It resembles an "Internet microscope", which lets us to look at a given network link, see the traffic flowing, and identify various types of IP

Paweł Foremski
The Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences,
Bałtycka 5, 44-100 Gliwice, Poland
E-mail: pjf@iitis.pl

Christian Callegari · Michele Pagano
Department of Information Engineering, University of Pisa,
Via Caruso 16, I-56122 Pisa, Italy
E-mail: {c.callegari,m.pagano}@iet.unipi.it

flows. Another useful metaphor to Traffic Classification (TC) is listening to two foreigners talking nearby and recognizing their human language. Quite often, we are able to identify an unfamiliar language or dialect even if we cannot fully understand it. Similarly, the TC problem is recognizing network protocols given their traffic, without interest in their full information content. Moreover, knowing the protocols behind IP flows makes networks easier to manage. For instance, TC is important for network monitoring: if we want to visualize the traffic flowing through a router, it is useful to know its components. TC also helps network security officers to reveal and track suspicious network activity. It is used for implementing Quality of Service (QoS) schemes, traffic shaping, and packet filtering. In convergent networks, TC is the mechanism that enables separate routing policies for voice, video, and data traffic.

A single IP packet alone is difficult to classify, as there is no application name in the packet headers. In the past, the service port number was used for discriminating the traffic class [2], but this became ineffective due to the raise of Peer-to-Peer (P2P) traffic in the early 2000s [3]. A popular and *de facto* standard method used nowadays is Deep Packet Inspection (DPI): pattern matching on full packet contents [4]. However, although being more accurate than port-based classification, it requires more computing power and brings privacy concerns. Moreover, pervasive encryption and other issues make DPI increasingly irrelevant [5, 6]. Instead, modern classifiers investigate groups of packets to find distinguishing features of specific application, rather than of single packets. Usually, a flow of packets is statistically summarized—for example, using the average packet size and inter-packet arrival time—and the resultant *feature vector* is classified using a Machine Learning (ML) algorithm [7]. Such methods are more reliable: the overall behavior of a particular protocol or host is examined instead of seeking for a strict match in a few packets.

The current challenge in TC is that in future it will have to deal with an increasing adoption of encryption, encapsulation, multi-channel techniques, and with the tremendous growth of the Internet [8]. Inevitably, the TC problem is becoming a very complex task that needs breaking into subproblems to keep it tractable. Recent papers proposed various interesting techniques tailored at subproblems in TC [9–11], but so far few authors addressed the problem of combining these proposals to work together. Thus, in this paper, we describe our method for integrating different traffic classifiers—the Waterfall architecture [12]—and we introduce a novel algorithm for optimizing such systems.

In more detail, we will show how to apply a Multiple Classifier Systems (MCS) technique called *cascade classification* [13] to build a modular TC system optimized for a given computer network. The Waterfall architecture lets for dedicated classifiers for different types of network traffic, thus we believe our contribution is important for convergent networks. For example, Long-Term Evolution (LTE) networks allow transmitting voice calls directly over IP, along with ordinary Internet data, which is known as Voice over LTE (VoLTE). Usually, the network will use a finite set of destination IP addresses for the VoLTE traffic. If one wants to identify IP traffic in such a network, Waterfall would allow separate classifiers for VoLTE traffic—which is simple, e.g. using the IP

address—and for typical Internet data—which is more challenging. In overall, the system would effectively use computing resources by applying adequate methods to various services present in the heterogeneous network. Moreover, our optimization technique would further tune the system for desired goal, e.g. real-time traffic visualization. Comparing to our introductory work [14], the contribution of this paper is as follows:

1. We give an extended description of our novel method for optimizing classification cascades (Sections 2 and 3).
2. We describe how to implement our algorithm recursively, and we reflect on its time complexity (Section 3.3).
3. We extensively validate our proposal on a new dataset with reliable ground-truth information, and on 7 classification modules total (Section 4).
4. We compare our proposal with myopic optimizer (Section 4.4).
5. We release an open source implementation of our proposal as a publicly available module (Section 5).

We begin our paper with Sections 2 and 3 describing the Waterfall architecture and our optimization method, respectively. Then, we present the experimental results in Section 4. We conclude our work in Section 5.

2 Cascade traffic classification

The field of network traffic classification needs a method for integrating results of various research activities. Many papers in this area describe classification methods that in principle propose a set of traffic features tailored at a set of network protocols [1, 9–11, 15–17]. 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 for traffic classification. 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.

In this Section we describe Waterfall: a modular architecture for traffic identification systems, which we introduced in [12]. Waterfall allows existing classification methods to complement each other, which makes the system as a whole capable of providing higher performance than could be achieved by any of the constituent modules.

2.1 Background

A naïve approach to the integration problem would be to survey recent papers for traffic features and use 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* [18]: 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 [10]. Some features need more packets to be computed: e.g. port number is available after one packet, whereas payload statistics need several tens of packets [11]. 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 difficult to develop and maintain.

Instead, researchers adopt multi-classification—in particular the Behavior Knowledge Space (BKS) combination method that fuses the 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 learning 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 [13], pp. 128). This powerful technique can increase the performance of TC systems—as shown by Dainotti et al. [19]—but comparing to Waterfall, it inherently requires *all* modules to be run on each flow, with the drawback that the more modules are used, the more processing power is required.

2.2 The Waterfall architecture

Waterfall applies the idea of multi-classification, but queries the constituent classifiers in sequential manner instead of parallel. It employs *cascade classification*, of which L. Kuncheva writes in her book on multi-classification: "cascade classifiers seem to be relatively neglected although they could be of primary importance for real-life applications." (in [13], pp. 106). We argue that cascade classification is a powerful and effective technique for combining algorithms that identify Internet traffic.

The Waterfall idea is presented in Figure 1. The input to the system is an IP flow—a feature vector $\mathbf{x}$—which contains all the features required by all 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,

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

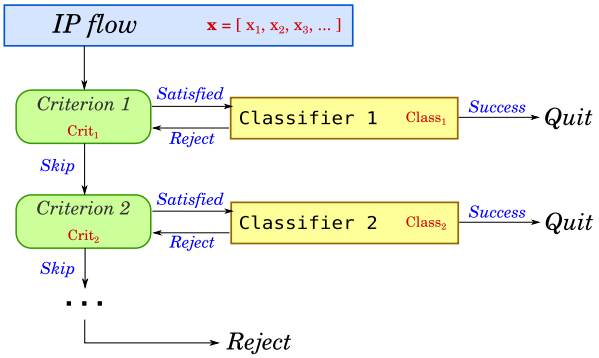

Fig. 1 The Waterfall architecture. A flow enters the system and is sequentially examined by the modules. In case of no successful classification, it is rejected.

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

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.

The selection criteria are designed to skip ineligible classifiers quickly. For example, in order to implement a module that identifies traffic by analyzing the packet payload sizes, the criterion could check if at least 5 packets with payload data were already sent in each direction. Only if this condition is true, a machine learning algorithm is run to identify the protocol. However, probably a large amount of flows will be skipped, saving computing resources and avoiding classification with an inadequate method. On the other hand, if a flow satisfies this criterion, it will be analyzed with a method that does not need to support corner cases (that is, number of payload packets less than 5). The selection criteria are optional, i.e. if a module does not have an associated criterion, the classification is always run.

3 Waterfall optimization

Now we will consider the problem of optimal cascade structure. Let F be a set of IP flows, and E be a set of n classification modules,

$$E = \{1, \ldots, n\} \tag{3}$$

that we want to use for cascade classification of flows in F in an optimal way. In other words, we need to find a sequence of modules X,

$$X = (x_1, \ldots, x_m) \quad m \leq n,\ x_i \in E,\ x_i \neq x_j \ \textit{for}\ i \neq j \tag{4}$$

that minimizes a cost function C,

$$C(X) = f(T_X) + g(E_X) + h(U_X) \tag{5}$$

where the terms T_X, E_X, and U_X respectively represent the total amount of CPU time used, the number of errors made, and the number of flows left unlabeled while classifying F with X. The terms f, g, and h denote arbitrary real-valued functions. Because $m \leq n$, some modules may be skipped in the optimal cascade. Note that U_X does not depend on the order of modules, because unrecognized flows always traverse till the end of the cascade.

3.1 Background

Cascade classification is a multi-classifier system implementing the classifier selection idea [13]. Interestingly, although first introduced in 1998 by E. Alpaydin and C. Kaynak [20], so far few authors considered the puzzle of optimal cascade configuration that would match our problem. In a 2006 paper [21], K. Chellapilla et al. propose a cascade optimization algorithm that updates the rejection thresholds of the constituent classifiers. The authors apply an optimized depth first search to find the cascade that satisfies given constraints on time and accuracy. However, comparing with our work, the system does not optimize the module order. In another paper on this topic, published in 2008 by A. Sherif [22], the author proposes a greedy approach for building cascades: start with a generic solution and sequentially prepend a module that reduces CPU time. Comparing with our work, the approach does not evaluate all possible cascade configurations and thus can lead to suboptimal results. We will demonstrate this in Section 4 for an exemplary myopic optimizer.

Thus, we propose a new solution to the cascade classification problem, which is better suited for traffic classification than existing methods. Note that comparing with [21] we do not consider rejection thresholds as input values to the optimization problem. Instead, in case of classifiers with tunable parameters, one could consider the same module parametrized with different values as separate modules, and apply our technique as well. For instance, a Bayes classifier with rejection thresholds on the posterior probability of $0.5, 0.75, 0.90$ would be considered as 3 separate modules.

3.2 Proposed solution

To find the optimal cascade, we propose to approximate the performance of every possible X by calculating the performance of each module on the entire dataset and then smartly combining the results. Note that for an accurate solution one would basically need to run the full classification process for all permutations of all combinations in E. This would take S experiments, where

$$S = \sum_{i=1}^{n} \frac{n!}{(n-i)!} \approx e \cdot n! \tag{6}$$

which is impractical even for small n. On another hand, fully theoretical models of the cost function seem infeasible too, due to the complex nature of the cascade and module inter-dependencies.

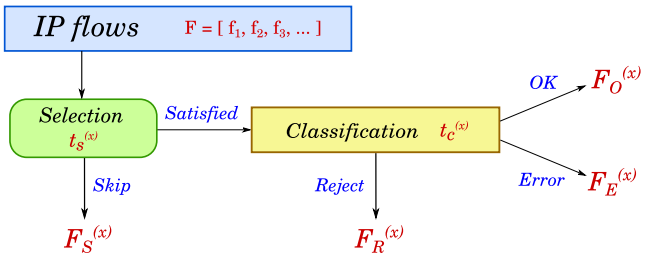

Fig. 2 Measuring performance of module $x \in E$.

Thus, we propose a heuristic solution to the cascade optimization problem. The algorithm has two evaluation stages:

A. Static: classify all flows in F using each module in E, and
B. Dynamic: find the X sequence that minimizes $C(X)$.

A. Static Evaluation. In every step of stage A, we classify all flows in F using each single module $x \in E$. We measure the average CPU time used for flow selection and classification: $t_s^{(x)}$ and $t_c^{(x)}$. We store each output flow identifier in one of the three outcome sets, depending on the result: $F_S^{(x)}$, $F_O^{(x)}$, or $F_E^{(x)}$. These sets hold respectively the flows that were skipped, properly classified, and improperly classified. Let us also introduce $F_R^{(x)}$,

$$F_R^{(x)} = F \setminus (F_S^{(x)} \cup F_O^{(x)} \cup F_E^{(x)}) \tag{7}$$

that is, the set of rejected flows. See Fig. 2 for an illustration of the module measurement procedure. As the result of every step, the performance of module x on F is fully characterized by a tuple $P^{(x)}$,

$$P^{(x)} = (\ F,\ t_s^{(x)}, t_c^{(x)}, F_S^{(x)}, F_O^{(x)}, F_E^{(x)}\ ) \tag{8}$$

Finally, after n steps of stage A, we obtain n tuples: a model of our classification system, which is the input to stage B.

B. Dynamic Evaluation. Having all of the required experimental data, we can quickly estimate $C(X)$ for arbitrary X. Because f, g, h, are used only for adjusting the cost function—and can be modified by the network administrator according to her needs (see Section 4.2)—we focus only on their arguments, i.e. the cost factors T_X, E_X, and U_X.

Let $X = (x_1, \ldots, x_i, \ldots, x_m)$ represent certain order and choice of modules, and G_i represent the set of flows entering the module number i,

$$G_1 = F \tag{9}$$

$$G_{i+1} = G_i \setminus (F_O^{(x_i)} \cup F_E^{(x_i)}) \quad 1 \le i \le m \tag{10}$$

then we estimate the cost factors using the following procedure:

$$T_X \approx \sum_{i=1}^{m} |G_i| \cdot t_s^{(x_i)} + |G_i \setminus F_S^{(x_i)}| \cdot t_c^{(x_i)} \tag{11}$$

$$E_X = \sum_{i=1}^{m} |G_i \cap F_E^{(x_i)}| \tag{12}$$

$$U_X = |G_{m+1}| \tag{13}$$

where $|G|$ denotes the number of flows in set G.

Note that the difference operator in Eq. 10 connects the static cost factors with the dynamic effects of a cascade. In stage A, our algorithm evaluates static performance of every module on the entire dataset F, but in stage B we want to simulate cascade operation, so we need to remove the flows that were classified in the previous steps. Thus, the operation in Eq. 10 is crucial.

Module performance depends on its position in the cascade, because preceding modules alter the distribution of traffic classes in the flows conveyed onward. For example, we can improve accuracy of a port-based classifier by putting a module designed for P2P in front of it, which should handle the flows that misuse the traditional port assignments.

3.3 Discussion

In our solution, instead of $e \cdot n!$ experiments (see Eq. 6), we simplified the optimization problem to n experiments and several computations, which in overall is much faster. Note that in case of adding a new module x_j to an already simulated cascade X, we can re-use previous computations:

$$G_j = U_X \tag{14}$$

$$T_{X+x_j} \approx T_X + |G_j| \cdot t_s^{(x_j)} + |G_j \setminus F_S^{(x_j)}| \cdot t_c^{(x_j)} \tag{15}$$

$$E_{X+x_j} = E_X + |G_j \cap F_E^{(x_j)}| \tag{16}$$

$$U_{X+x_j} = G_j \setminus (F_O^{(x_j)} \cup F_E^{(x_j)}) \tag{17}$$

Thus, we suggest searching for the minimum $C(X)$ in a recursive algorithm. However, although simulation is orders of magnitude faster than experimentation, we still check every possible cascade. This makes the time complexity of our algorithm factorial, considering set computations as the elementary operations. This might leave space for further improvements by the introduction of heuristics, possibly tuned to a specific cost function.

Moreover, note that the results depend on F: the optimal cascade depends on the protocols present in the traffic, and on the ground-truth labels. The presented method cannot provide the ultimate solution that would match every network, but it can optimize a specific cascade system for a specific network. We further discuss this issue in Section 4.

We assume that the flows are independent of each other, i.e. labeling a particular flow does not require information on any other flow. If such information

Dataset	Start	Duration	Src. IP	Dst. IP	Packets	Bytes	Avg. Util	Avg. Flows (/5 min.)	Payload
Asnet1	2012-05-26	216 h	1,800 K	1,500 K	2,500 M	1,600 G	18 Mbps	7.7 K	92 B
Asnet2	2013-01-24	168 h	2,500 K	2,800 K	2,800 M	1,800 G	26 Mbps	12 K	84 B
IITiS1	2012-05-26	216 h	32 K	46 K	150 M	95 G	1.0 Mbps	750	180 B
Unibs1	2009-09-30	58 h	27	1 K	30 M	26 G	0.9 Mbps	110	0 B
UPC1	2013-02-25	65 d	90 K	18 K	37 M	33 G	51 Kbps	68	full
	2013-11-18	35 d	7.5 K	54 K	43 M	31 G	88 Kbps	49	full

Table 1 Datasets used for experimental validation.

is needed, e.g. flow DNS names, it should be extracted before the classification process starts. Thus, traffic analysis and flow classification must be separated to uphold this assumption. We successfully implemented such systems for our DNS-Class [10] and Mutrics [12] classifiers.

In the next Section, we experimentally validate our method and show that it perfectly predicts E_X and U_X, and approximates T_X properly. The simulated cost follows the real cost, so we claim our proposal is valid and can be used in practice. We also analyze the trade-offs between speed, accuracy, and ratio of unlabeled flows, to stress out that the final choice of the cost function should depend on the purpose of the system.

4 Experimental validation

Below we present the outcome of using real traffic datasets for experimental evaluation of our proposal. We ran 4 experiments:

1. comparing simulation with reality, which proves validity of Eqs. 11-13;
2. analyzing the effect of cost function parameters on the result, which demonstrates optimization for different goals;
3. optimizing on one dataset and using the cascade on another dataset, which evaluates stability;
4. comparing our optimization method with myopic optimization, which shows that our work is meaningful.

For the experiments, in general we used 5 datasets, summarized in Table 1. Datasets Asnet1 and Asnet2 were collected at the same ISP serving <500 domestic users, with an 8-month time gap. Dataset IITiS1 was collected at an academic network serving <50 researchers, at the same time as Asnet1. Dataset Unibs1 was also collected at an academic network (University of Brescia[1]), but a few years earlier and using a reliable ground-truth information [23] (this dataset was anonymized). Finally, the UPC1 dataset was artificially generated—with manual simulation of different human behaviors—hence it contains full packet payloads and the names of applications that generated the traffic flows [24–26].

For the first 3 datasets, we established ground-truth using light DPI [27]. For Unibs1 and UPC1, we used the supplied ground-truth information, which

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

Module	ML algorithm	Traffic features
`dnsclass`	linear SVM	DNS name
`dstip`	lookup table	destination IP address
`npkts`	random forest	payload sizes: first 4 packets in+out
`port`	lookup table	destination port number
`portsize`	lookup table	payload sizes: first packet in+out
`portname`	lookup table	DNS name
`stats`	random forest	4 basic statistics of packet sizes and inter-arrival times

Table 2 Waterfall modules used for experimental validation.

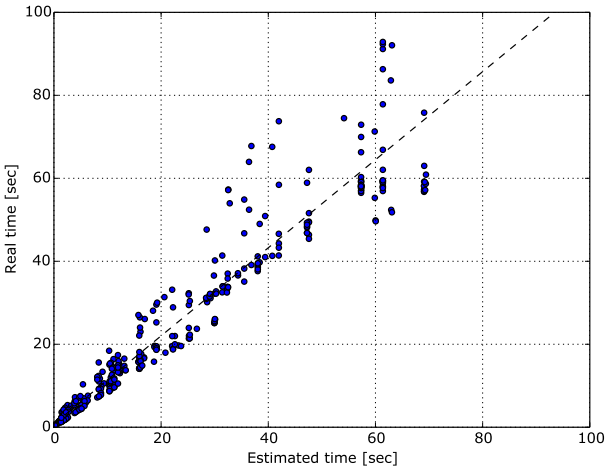

Fig. 3 Experiment 1. Estimated classification time vs real classification time. Dashed line shows least-squares approximation. The Pearson product-moment correlation is 0.95.

sometimes was challenging: for example, a `skype` process generates some HTTP traffic apart of the Skype protocol. For each dataset, we trained the modules using 60% random sample of all flows, and used the remainder for testing. We considered only the first 10 seconds of each flow to resemble a near-immediate traffic identification.

Finally, in total we evaluated 7 classification modules, summarized in Table 2 [10, 12]. As additional traffic features, we used the transport protocol and destination port number for each module. Although we consider port numbers as an unreliable feature, they still can provide valuable hint for more sophisticated classification mechanisms. Note that the modules support the *reject option*, so each module can drop any flow if its not certain about the outcome.

4.1 Experiment 1

In the first experiment, we compare simulated cost factors with real values for arbitrary cascade configurations. We randomly selected 100,000 flows from each of the first 4 datasets and ran static evaluation on them. Next, we generated 100 random cascades, and for each cascade we ran both real and sim-

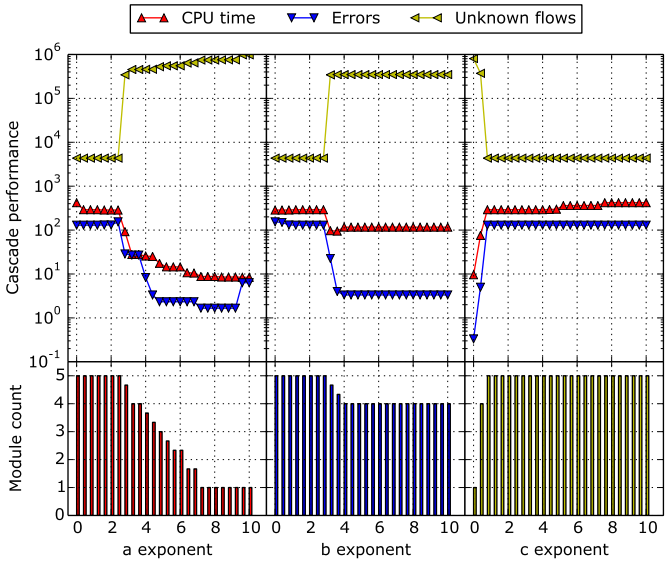

Fig. 4 Experiment 2. Optimizing the cascade for different goals: best classification time (a exponent), minimal number of errors (b exponent), and the lowest number of unlabeled flows (c exponent): the plot shows the averages for 3 datasets.

ulated classification. As a result, we obtained corresponding pairs of real and estimated values of T_X, E_X, and U_X.

The results for T_X are presented in Fig. 3. For E_X and U_X we did not observe a single error, i.e. our method perfectly predicted the real values. For CPU time estimations, we see a high correlation of 0.95, with little underestimation of the real value. For all datasets, the estimation error was below 20% for majority of evaluated cascades (with respect to the real value). The error was above 50% only for 5% of evaluated cascades.

We conclude that in general our method properly estimates the cost factors and we can use it to simulate different cascade configurations. Note that accurate prediction of the CPU time is not necessary for optimization: it is enough for the simulated time to be roughly proportional to the real value. Moreover, even the real values will vary depending e.g. on the CPU load due to other tasks executed in the background, which is difficult to predict.

4.2 Experiment 2

In our second experiment we show the effect of tuning the system for 3 different goals: a) minimizing the computation time, b) minimizing errors, and c) labeling as many flows as possible. We chose the following cost function:

$$C(X) = f(T_X) + g(E_X) + h(U_X) = (T_X)^a + (E_X)^b + (U_X)^c \qquad (18)$$

with the default values of a, b, c equal to $0.95, 1.75, 1.20$, respectively. We separately varied these values in range of 0-10, and observed the performance of the resultant cascades. For the sake of brevity, we ran the experiment for

Reference	Test dataset			
	Asnet1	*Asnet2*	*IITiS1*	*UPC1*
Asnet1		1.01%	5.31%	48.96%
Asnet2	2.67%		7.29%	23.34%
IITiS1	33.37%	34.19%		192.91%
UPC1	14.51%	11.11%	31.77%	

Table 3 Experiment 3. Result stability: relative increase in the cost $C(X)$, depending on the reference dataset used for determining the optimal cascade.

datasets ASNET1, ASNET2, and IITiS1 and for modules `dnsclass`, `dstip`, `npkts`, `port`, and `portsize`.

In Fig. 4, we present the results: dependence of cascade performance and module count on the cost function parameters. As expected, higher a exponent leads to faster classification and usually less errors, but with fewer modules in the cascade, and more unclassified flows as a consequence. Optimizing for accuracy—higher b exponent—leads to reduction of errors at the cost of higher number of flows left without a label. Finally, if we choose to classify as much traffic as possible (increasing the c exponent), the system will use all available modules, at the cost of higher CPU time and error rate.

In more detail, for time optimization, the *optimal* cascades are: `port` for ASNET1, `portsize` for ASNET2, and `dnsclass` for IITiS1. In the last case, `dnsclass` is preferred due to high percentage of DNS traffic in IITiS1. Instead, in case of accuracy optimization, the *optimal* cascades are: `portsize, dnsclass, npkts, port` for ASNET1, `dstip, dnsclass, portsize` for ASNET2, and `dnsclass, port, dstip, portsize, npkts` for IITiS1. Finally, optimizing for minimum percentage of unrecognized flows yields a common result for all datasets: `dnsclass, dstip, npkts, port, portsize`.

Note that the results depend on the cost function. We used a power function for presentation purposes, in order to easily show contrasting scenarios by small adjustments to the exponents. For specific purposes, a multi-linear function may be more appropriate, as it is often found in the literature, e.g. linear scalarization of multi-objective optimization problems. Moreover, more complex expressions—including thresholds on some parameters—can be used to find a classification system capable of real-time operation: given an expected amount of flows per second, one could find a cascade that is fast enough to handle the traffic while keeping the other cost factors at possible minimum.

We conclude that our proposal works and is adaptable, i.e. by varying the parameters we optimized the classification system for different goals.

4.3 Experiment 3

In the third experiment, we wanted to verify if the result of optimization is stable in time and space, i.e. if the optimal cascade stays optimal with time and changes of the network. We ran our optimization procedure for 4 datasets, obtaining different cascade configuration for each dataset. Next, we evaluated these configurations on all datasets and measured the increase in the cost

Dataset	Algorithm	Cascade configuration	Time [s]	Errors	Unknowns
Asnet1	myopic	portname,portsize,port,dstip,dnsclass,stats,npkts	89	40	886
	optimal	portsize,portname,dstip,dnsclass,npkts,port,stats	87	30	886
			+2%	**+26%**	**0%**
Asnet2	myopic	portname,portsize,port,dstip,dnsclass,stats,npkts	141	49	817
	optimal	portsize,portname,dstip,dnsclass,npkts,port	139	22	1,224
			+2%	**+55%**	**-50%**
IITiS1	myopic	dnsclass,port,portname,portsize,dstip,stats,npkts	5.7	2.4	80
	optimal	port,portsize,npkts,stats	5.1	2.4	80
			+11%	**+0%**	**0%**
Unibs1	myopic	portsize,port,dstip,stats,npkts	102	2,017	13,892
	optimal	dstip,portsize,port,npkts,stats	91	1,985	13,892
			+10%	**+2%**	**0%**
UPC1	myopic	portname,port,portsize,dstip,dnsclass,stats,npkts	110	686	1,746
	optimal	port,portname,dstip,portsize,dnsclass,npkts,stats	92	604	1,746
			+16%	**+12%**	**0%**
		Average improvement:	**+8%**	**+19%**	**-10%**

Table 4 Experiment 4. Average improvements compared to myopic cascade optimization.

function $C(X)$ compared with the original value. Note that we did not use the UNIBS1 dataset for this experiment, as it lacks packet payloads and hence needs different set of available modules.

Table 3 presents the results. We see that our proposal yielded results that are stable in time for the same network: the cascades found for ASNET1 and ASNET2, which are 8 months apart, are similar and can be exchanged with little decrease in performance. However, the cascades found for ASNET1 and ASNET2 gave 5-7% worse performance compared with IITiS1, and 23-49% worse performance on UPC1. We observed extreme decrease in performance when we varied both the network and time, especially when classifying UPC1 with cascade optimized for IITiS1.

We conclude that cascade optimization is specific to the network, but on the other hand our results suggest that an optimal cascade does not change significantly with time for given network. Thus, the network administrator does not need to repeat the optimization procedure frequently.

4.4 Experiment 4

In the last experiment, we compared our proposal with a greedy optimizer, i.e. a situation in which we select all modules in order of increasing CPU time. This resembles the basic approach in the original paper on Waterfall [12]: start with generic, heavy classifier, and prepend faster modules in front of it (see Section 5 in [12]). Thus, for each module, we calculated the sum of t_s and t_c for each dataset separately, and ordered the modules from the fastest to the slowest. We used the results as cascade configurations, i.e. Waterfall systems configured with a conservative algorithm: "myopic" optimization.

On the other hand, we also optimized the system using our proposal, with the cost function given in Eq. 18, for a, b, c equal to $3.00, 1.75, 1.50$, respectively. We chose these exponent values arbitrarily to show an example of time optimization: note that the a exponent (influencing the time cost factor) is the

highest. Then, we used the results as cascade configurations, but optimized with an "optimal" algorithm.

Table 4 compares the results: in every case, our algorithm optimized the classification system to work faster and with less errors, usually with the same amount of unclassified flows. This demonstrates the point of cascade optimization: it brings performance improvements. Recall that UNIBS1 lacks packet payloads, hence we used 5 modules in general for this dataset instead of 7.

On average, the system worked 8% faster compared with myopic time optimization, and reduced the error rate by 19%. For ASNET2, it also resulted in higher number of unrecognized flows, but the increase is insignificant given the dataset size, and this cost factor was not the goal of optimization. For instance, if one wants a real-time traffic visualization system, then some small portion of flows might remain unrecognized without negative effect on the whole system. Thus, we conclude that our work is meaningful and can help network administrators to tune cascade TC systems better than ad-hoc tools.

5 Conclusions

We showed that our Waterfall architecture, together with the new optimization technique, lets for effective combining of traffic classifiers. We presented background on cascade classification (a multi-classifier variant) and employed it for identifying IP transmissions. Waterfall brakes the complex TC problem into smaller, independent modules, which are easier to manage. Moreover, we presented an optimization technique that automatically selects the set of best modules from a pool of available methods, and puts them in right order for maximized performance. By means of experimental validation we demonstrated that our proposal works and can bring significant improvements to classification speed, accuracy, and number of recognized flows.

Our approach to optimizing Waterfall systems brings major improvements over ad-hoc methods. First, it reduces the time needed for optimization by orders of magnitude, by replacing experimentation on different cascades with simulation, which is much faster. Second, by performing an exhaustive search for the best solution, it finds better cascades than a greedy algorithm. However, due to the complex nature of the problem, it still requires a considerable amount of computations to check for all possible cascade configurations, which in practice limits the maximum size of the module pool.

We believe our contribution is important for managing convergent networks like LTE. Finally, in order to support further research in this area, we release an open source implementation of our proposal as an extension to the MUTRICS classifier, available at `https://github.com/iitis/mutrics`.

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. Keys, K., Moore, D., Koga, R., Lagache, E., Tesch, M., Claffy, K.: The architecture of CoralReef: an Internet traffic monitoring software suite. In: PAM2001, Workshop on Passive and Active Measurements, RIPE, Citeseer (2001)
3. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K.C., Faloutsos, M.: Is P2P dying or just hiding? In: Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE. Volume 3., IEEE (2004) 1532–1538
4. Sen, S., Spatscheck, O., Wang, D.: Accurate, scalable in-network identification of P2P traffic using application signatures. In: Proceedings of the 13th international conference on World Wide Web, ACM (2004) 512–521
5. Dusi, M., Gringoli, F., Salgarelli, L.: Quantifying the accuracy of the ground truth associated with Internet traffic traces. Computer Networks **55**(5) (2011) 1158 – 1167
6. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: Blinc: multilevel traffic classification in the dark. In: ACM SIGCOMM Computer Communication Review. Volume 35., ACM (2005) 229–240
7. Kim, H., Claffy, K.C., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: Internet traffic classification demystified: Myths, caveats, and the best practices. In: Proceedings of the 2008 ACM CoNEXT conference, ACM (2008) 11
8. Dainotti, A., Pescape, A., Claffy, K.C.: Issues and future directions in traffic classification. Network, IEEE **26**(1) (2012) 35–40
9. Bermolen, P., Mellia, M., Meo, M., Rossi, D., Valenti, S.: Abacus: Accurate behavioral classification of P2P-TV traffic. Computer Networks **55**(6) (2011) 1394 – 1411
10. Foremski, P., Callegari, C., Pagano, M.: DNS-Class: immediate classification of IP flows using DNS. International Journal of Network Management **24**(4) (2014) 272–288
11. Finamore, A., Mellia, M., Meo, M., Rossi, D.: KISS: Stochastic packet inspection classifier for udp traffic. Networking, IEEE/ACM Transactions on **18**(5) (2010)
12. Foremski, P., Callegari, C., Pagano, M.: Waterfall: Rapid identification of IP flows using cascade classification. In: Computer Networks. Springer (2014) 14–23
13. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley (2004)
14. Foremski, P., Callegari, C., Pagano, M.: Waterfall traffic identification: optimizing classification cascades. In: Computer Networks. Springer (2015) 1–10
15. Fiadino, P., Bär, A., Casas, P.: HTTPTag: A Flexible On-line HTTP Classification System for Operational 3G Networks. In: International Conference on Computer Communications, 2013. INFOCOM'13, IEEE (2013)
16. 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) (2012) 386–403
17. Korczynski, M., Duda, A.: Markov chain fingerprinting to classify encrypted traffic. In: INFOCOM, 2014 Proceedings IEEE, IEEE (2014)
18. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern classification. John Wiley & Sons (2012)
19. Dainotti, A., Pescapé, A., Sansone, C.: Early classification of network traffic through multi-classification. Traffic Monitoring and Analysis (2011) 122 – 135
20. Alpaydin, E., Kaynak, C.: Cascading classifiers. Kybernetika **34**(4) (1998) 369–374
21. Chellapilla, K., Shilman, M., Simard, P.: Optimally combining a cascade of classifiers. In: Proceedings of SPIE. Volume 6067. (2006) 207–214
22. Abdelazeem, S.: A greedy approach for building classification cascades. In: Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on, IEEE (2008) 115–120
23. Gringoli, F., Salgarelli, L., Dusi, M., Cascarano, N., Risso, F., Claffy, K.: Gt: Picking up the truth from the ground for internet traffic. ACM SIGCOMM Computer Communication Review **39**(5) (2009) 13 – 18
24. Bujlow, T., Carela-Español, V., Barlet-Ros, P.: Independent comparison of popular DPI tools for traffic classification. Computer Networks **76** (2015) 75–89
25. Carela-Español, V., Bujlow, T., Barlet-Ros, P.: Is our ground-truth for traffic classification reliable? In: Passive and Active Measurement, Springer (2014) 98–108
26. Carela-Español, V., Barlet-Ros, P., Cabellos-Aparicio, A., Solé-Pareta, J.: Analysis of the impact of sampling on NetFlow traffic classification. Computer Networks **55**(5) (2011) 1083–1099
27. Alcock, S., Nelson, R.: Libprotoident: Traffic classification using lightweight packet inspection. WAND Network Research Group, Tech. Rep (2012)