UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

PH.D. THESIS

IN

INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

METHODOLOGIES FOR MOBILE AND ENCRYPTED TRAFFIC CLASSIFICATION VIA MACHINE LEARNING APPROACHES

ANTONIO MONTIERI

TUTOR: PROF. ANTONIO PESCAPÈ

COORDINATOR: PROF. DANIELE RICCIO

XXXII CICLO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Corso di Dottorato di Ricerca in
Information Technology and Electrical Engineering
XXXII Ciclo
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

METHODOLOGIES FOR MOBILE AND ENCRYPTED TRAFFIC CLASSIFICATION VIA MACHINE LEARNING APPROACHES

ANTONIO MONTIERI

Ph.D. Thesis

TUTOR
Prof. Antonio Pescapé

COORDINATOR
Prof. Daniele Riccio

March 2020

Ad Angela.

Abstract

The widespread use of handheld devices (*e.g.*, smartphones) has led to a significant evolution in the way the users connect to the Internet and access contents or services. This entails a substantial change in the nature of network traffic. Traffic classification—the set of techniques suited to infer the applications generating network traffic—is currently the enabler for gathering valuable information for different stakeholders in the Internet traffic delivery supply chain. This includes its application for network management (*e.g.*, service differentiation/blocking and quality-of-service enforcement), network security, and user profiling. On top of that, traffic classification highlights compelling privacy issues related to (the share of) this information in thorny scenarios (*e.g.*, healthcare apps and enterprise environments).

Nonetheless, the proliferation of encryption (*e.g.*, anonymity tools) hinders the suitability of solutions based on cleartext traffic inspection and thus challenges current classifiers. Also, the moving-target nature of mobile traffic, due to the daily-expanding set of apps sharing common third-party services, accelerates the performance degradation of design solutions based on standard machine learning approaches.

As such, this Thesis presents a set of novel methodologies for mobile traffic classification that can operate under the encrypted-traffic assumption and advances the state-of-the-art from multiple viewpoints. In detail, the present dissertation devises innovative machine learning approaches based on multi- and hierarchical-classification. Furthermore, it pioneers the adoption of the deep learning paradigm to design practical and effective mobile traffic classifiers through the automatic extraction of features reflecting complex data patterns. Then, to overcome the complexity of these solutions, a distributed deployment based on the big-data framework is investigated.

Such analysis highlights the non-transparent nature of the big-data accelerator when applied to the training phase of deep learning classifiers, shedding light on intrinsic trade-offs. Extensive experimental evaluations are conducted to assess the performance of proposed approaches and compare them with most related state-of-the-art solutions. This goal is achieved by the definition of a common benchmark encompassing public datasets. In this regard, a novel architecture is designed and implemented to capture and label our publicly-released dataset.

Contents

List of Publications

The work presented in this Thesis resulted in the publication of the following papers during my three-year Ph.D. programme.[1] All the authors equally contributed to each publication, regardless of the authorship order. Specifically, for each publication, I was actively involved in all the phases of the research activity and writing of the manuscript.

International Journal Publications

1. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "MIMETIC: Mobile Encrypted Traffic Classification using Multimodal Deep Learning", Elsevier Computer Networks (COMNET), Volume 165, 24 December 2019.

2. **Antonio Montieri**, Domenico Ciuonzo, Giampaolo Bovenzi, Valerio Persico, Antonio Pescapé, "A Dive into the Dark Web: Hierarchical Traffic Classification of Anonymity Tools", IEEE Transactions on Network Science and Engineering (TNSE), 2019, Early Access.

3. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges", IEEE Transactions on Network and Service Management (TNSM), Volume 16, Number 2, June 2019, Pages 445–458.

[1] It is worth noting that the present list is not a comprehensive report of my publication track but it encompasses only the papers concerning the main subject of this Thesis.

4. **Antonio Montieri**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Pescapé, "Anonymity Services Tor, I2P, JonDonym: Classifying in the Dark (Web)", IEEE Transactions on Dependable and Secure Computing (TDSC), 2018, Early Access.

5. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "Multi-Classification Approaches for Classifying Mobile App Traffic", Elsevier Journal of Network and Computer Applications (JNCA), Volume 103, 1 February 2018, Pages 131–145.

6. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "Toward Effective Mobile Encrypted Traffic Classification through Deep Learning", Submitted to Elsevier Neurocomputing (NEUCOM), Under Review.

International Conference and Workshop Publications

1. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Valerio Persico, Antonio Pescapé, "MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation", 4th IEEE International Conference on Computing, Communications and Security (ICCCS 2019), October 10–12, 2019, Rome, Italy.

2. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Valerio Persico, Antonio Pescapé, "Know your Big Data Trade-offs when Classifying Encrypted Mobile Traffic with Deep Learning", 3rd Network Traffic Measurement and Analysis Conference (TMA 2019), June 17–21, 2019, Paris, France.

3. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "Mobile Encrypted Traffic Classification Using Deep Learning", 2nd Network Traffic Measurement and Analysis Conference (TMA 2018), June 26–29, 2018, Vienna, Austria.

4. Giuseppe Aceto, Domenico Ciuonzo, **Antonio Montieri**, Antonio Pescapé, "Traffic Classification of Mobile Apps through Multi-classification", 2017 IEEE Global Communications Conference (IEEE GLOBECOM 2017); Communication QoS, Reliability and Modeling (CQRM) Symposium, December 4–8, 2017, Singapore.

5. **Antonio Montieri**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Pescapé, "Anonymity Services Tor, I2P, JonDonym: Classifying in the Dark", 29th International Teletraffic Congress (ITC 29), September 4–8, 2017, Genova, Italy.

List of Figures

List of Tables

List of Acronyms

ADB Android Debug Bridge.

AE AutoEncoder.

AEASGD Asynchronous Elastic Averaging Stochastic Gradient Descent.

AT Anonymity Tool.

BD Big Data.

BKS Behavior-Knowledge Space.

BN Bayesian Network.

BT Burst Threshold.

CART Classification And Regression Tree.

CC Class-Conscious.

CDN Content Delivery Network.

CI Class-Indifferent.

CNN Convolutional Neural Network.

CR Classified Ratio.

DL Deep Learning.

DP Decision Profile.

DPI Deep Packet Inspection.

DS Dempster-Shafer.

DT Decision Templates.

ECE Expected Calibration Error.

FB Facebook.

FBM Facebook Messenger.

FI Fuzzy Integral.

FNR False Negative Rate.

FPR False Positive Rate.

GRU Gated Recurrent Unit.

GT Ground Truth.

HC Hierarchical Classification.

HMM Hidden Markov Model.

I2P Invisible Internet Project.

IAT Inter-Arrival Time.

IoT Internet of Things.

ISP Internet Service Provider.

K-NN K-Nearest Neighbors.

LCL Local Classifier per Level.

LCN Local Classifier per Node.

LCPN Local Classifier per Parent Node.

LSTM Long Short-Term Memory.

MC Multi-Classification.

MCS Multi-Classification System.

MIOBC Maximum Improvement Over Best Classifier.

ML Machine Learning.

MLP MultiLayer Perceptron.

MNB Multinomial Naïve Bayes.

MV Majority Voting.

NAT Network Address Translation.

NB Naïve Bayes.

ORA Oracle Combiner.

P2P Peer-to-Peer.

PCA Principal Component Analysis.

PL Packet Length.

PP Probabilistic Product.

PT Pluggable Transport.

REC Recall Combiner.

RF Random Forest.

RQ Research Question.

RTPE Run-Time Per-Epoch.

SAE Stacked AutoEncoder.

SB Service Burst.

SOA Soft-Output Average.

SVC Support Vector Classifier.

SVM Support Vector Machine.

TC Traffic Classification.

TI Traffic Identification.

TLF "Trainable" Late Fusion.

TNR True Negative Rate.

TOR The Onion Router.

TPR True Positive Rate.

VPN Virtual Private Network.

WER Wernecke's method.

WF Website Fingerprinting.

WMV Weighted Majority Voting.

WTPE Wallclock Time Per-Epoch.

Chapter 1

Introduction and Background

Internet is the most crucial technology of the current age, having a tremendous impact on economics, society, and politics with a user base of more than 4.3 billion people, encompassing the 57% of total population around the globe [1, 2]. However, the original design principles of the Internet [3] were based on the best-effort paradigm and did not foresee the present development as a global network of networks and the need for functionalities not included in the initial Internetting concepts [4]. These needs include, inter alia, security, privacy, uninterrupted access, and quality-of-service guarantees. They are provided by means of various tools, such as security or quality-of-service enforcement devices and network monitors, that are in charge of managing, prioritizing or blocking certain network traffic. Since these tools base their operations on the knowledge of the network traffic, their use is limited (or impaired) when this requirement is not (or loosely) satisfied. Thus, network traffic analysis (*i.e.* the umbrella of procedures for distilling information from network traffic) represents an activity of paramount importance that must follow and adapt to the fast-paced

traffic evolution.

In this chapter, Sec. 1.1 describes the mobile and encrypted context in which the present study is carried out, underlining also its motivations. Thereafter, Sec. 1.2 introduces the key subjects recently tackled in mobile traffic analysis and Sec. 1.3 deepens the basic concepts underlying network traffic classification. The last two sections illustrate the contribution of the Thesis along with an overview on the most related literature (§1.4) and outline its structure (§1.5).

1.1 The Growth of Mobile Traffic

The growing usage of smartphones in everyday life is deeply (and rapidly) changing the nature of traffic traversing home and enterprise networks, and the Internet. Smartphones have become the main medium of communication, providing fast and almost ubiquitous means for connecting groups of users as well as users to services. Due to the users' massive shift toward mobile devices and mobile applications (in short, apps) running on them, overall network traffic reached huge volumes and started evolving at an unprecedented pace. Such rate of change is further increased for mobile apps due to the software distribution systems (*i.e.* the app marketplaces), that have fostered one-click installation and quick-paced automatic updates.

According to the June 2019 Ericsson mobility report [5], in the last year (*i.e.* between the first quarter of 2018 and 2019), mobile data traffic has grown 82%, being fueled by both the rising number of smartphone subscriptions and the increasing average data volume per subscription. Moreover, the number of smartphone subscriptions is expected to reach 7.2 billions by 2024, with a corresponding +30% predicted compound annual growth rate of the traffic generated by mobile networks. Figure 1.1 reports the details

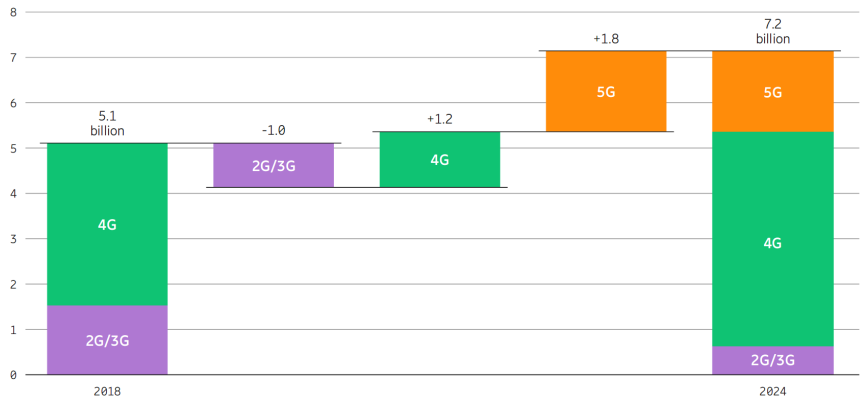


Figure 1.1: Forecast of smartphone subscriptions by technology (in billions). Source: [5].

of this forecast, showing also a shift toward higher-throughput technologies (namely +1.2 and +1.8 billions for 4G and 5G, respectively) that will contribute to the increase of mobile traffic data against broadband fixed connections, whose subscriptions are expected to show a very limited growth of around 3% per year through 2024. Interestingly, the proliferation of traffic-demanding apps is also expected to nourish the mobile traffic growth, with mobile video traffic accounting nearly the 75% of mobile data traffic, from about the current 60%. Notably, the results from the 2019 Sandvine's Mobile Internet Phenomena Report [6] also deepen this analysis, showing that YouTube is the global mobile apps' leader with over 35% of worldwide mobile traffic.

The huge increment of mobile data traffic has promoted great interest in mobile traffic analysis. Indeed, analyzing the traffic of mobile apps has the potential of providing extremely valuable profiling information that could be useful for a plethora of stakeholders, like advertisers, insurance and healthcare companies, and security agencies. On the other hand, it surely raises privacy issues, with the users having no insights about the informa-

tion that is leaked from mobile data they generate, neither about who will use this information and for what end. This is even exacerbated for sensitive apps, as children-compliant apps [7] or context-sensitive apps (such as health and dating ones) [8], and in controlled environments (*e.g.*, companies enforcing bring-your-own-device policy), or when malicious parties can infer and leverage mobile-traffic knowledge.

The next section gives a sketch of the recent subjects faced with the aid of mobile traffic analysis.

1.2 Mobile Network Traffic Analysis

Monitoring the characteristics and behavior of networks has long been a crucial task for operators, researchers, and developers. Telecom operators and Internet Service Providers (ISPs) have a long history of traffic-data analysis operations, possess a huge availability of network-level data, and have thus enjoyed decades-long research and applications on the topic.

This section provides a brief review of (mobile) network traffic analysis, discussing the key *subjects* tackled in last years, as summarized in Tab. 1.1. For each subject, the latter highlights the related privacy (P), security (S), and network management (M) *concerns* (possibly even partially affected by the considered subject), along with the *inference task* associated (*i.e. time-series prediction* or *binary/multi-class classification*). Moreover, it lists a few exemplifying *papers* together with the *methods* applied as design solutions to solve the related task. It is worth noting that the focus is on the most recent papers (*i.e.* published within the last 5 years) to show also the latest advancement of the techniques employed in mobile traffic analysis.[1]

[1]Traffic identification and classification are excluded, since a dedicated overview is provided in Sec. 1.3.

Table 1.1: Taxonomy of subjects pertaining to mobile network traffic analysis.

Subject	Concerns P	S	M	Inference Task	Method	Paper
Network Prediction	○	○	●	Time-series prediction	MLP, SVM	Nikravesh *et al.* [9]
					CNN+LSTM	Zhang *et al.* [10]
					DBN	Nie *et al.* [11]
Anomaly Detection	○	●	◐	Binary classification	K-Means, HC	Parwez [12]
					ANN	R.-Grammatikis, *et al.* [13]
					CNN	Marín, *et al.* [14]
Attack Classification	○	●	●	Multi-class classification	SAE	Thing [15]
					AE, RF	Sridharan *et al.* [16]
					LSTM	Diro, *et al.* [17]
Malware Detection	◐	●	◐	Binary classification	NB	Arora *et al.* [18]
					DNN	Chen *et al.* [19]
					SVM, C4.5 NB, BN, AB	Chen *et al.* [20]
Malware Classification	◐	●	●	Multi-class classification	CNN	Wang *et al.* [21]
					RF, RT, C4.5 KNN, LR	Lashkari *et al.* [22]
					CNN	Huang *et al.* [23]
Website Fingerprinting	●	○	○	Multi-class classification	JI	Spreitzer *et al.* [24]
					AE, CNN, LSTM	Rimmer *et al.* [25]
					AE, CNN	Sirinam *et al.* [26]
Traffic Identification	●	◐	◐	Binary classification	Analysis in Section 1.3	
Traffic Classification	●	◐	●	Multi-class classification		

Concerns: Privacy (**P**), Security (**S**), Management (**M**).
ML-Method: AdaBoost (AB), Bayesian Network (BN), Hierarchical Clustering (HC), K-Nearest Neighbors (KNN), Linear Regression (LR), Naïve Bayes (NB), Random Forest (RF), Random Tree (RT), Support Vector Machine (SVM).
DL-Method: AutoEncoder (AE), ANN (Artificial Neural Network) Convolutional Neural Network (CNN), Deep Belief Network (DBN), Deep Neural Network (DNN), Long Short-Term Memory (LSTM), MLP (MultiLayer Perceptron), SAE (Stacked AutoEncoder).
Other Method: JI (Jaccard Index). "+" symbol indicates hybrid methods.

In this regard, from Tab. 1.1, it can be noticed that the vast majority of most recent works in mobile traffic analysis employ methods based on both supervised and unsupervised Machine Learning (ML) and Deep Learning (DL) (cf. §1.3.4). Indeed, the huge success of ML/DL in several fields has recently ignited global interests in exploiting these paradigms also in networking [27, 28], where their adoption can leverage this solid know-how and help facing new challenges of mobile network-level data analysis.

The present taxonomy is not strictly tight, since some degree of overlapping could be possible between certain works on related subjects. A description of these subjects is given hereinafter. For example, studies tackling malware classification usually also perform malware detection, as a preliminary step of their analysis. Moreover, malware and (normal) traffic classification have been also investigated together, as in W. Wang *et al.* [21] and H. Huang *et al.* [23], both as separate problems or in a multi-task fashion, respectively.

Network Prediction. It refers to forecasting network traffic or performance indicators given historical measurements or related data. Specifically for mobile networks, given the high variability of both traffic and network conditions, as well as the stringent QoS requirements of new applications, this constitutes a challenging subject. Hence, the design of algorithmic solutions with increased traffic prediction abilities directly reflects on improved network management.

Anomaly Detection and Attack Classification. The aim is to reveal anomalies in the traffic due to attacks (*anomaly detection*) based on patterns drawn from normal network behavior, and, possibly, to infer also the specific attack experienced (*attack classification*). Specifically, the attacks

against mobile networks are usually based on well-known vulnerabilities (*e.g.*, location leaks and denial of service in LTE). Accordingly, both these subjects are directly linked to the security aspect, whereas attack classification allows a finer network management, for example attack-specific network countermeasures.

Malware Detection and Classification. The aim of *malware detection* is to identify whether the observed network traffic is generated by either legitimate apps or malware, while *malware classification* also tries to infer the malware type. In the mobile domain, these subjects are even more urgent since the countermeasures against malware are based on warning the users on the permissions given to a certain app, being ineffective when malware and benign apps require the same permissions. Hence, these subjects both pertain to the security aspect. Besides, privacy aspects are involved when malware provokes *data exfiltration*, while the network management aspect is partially (resp. fully) affected by advances in malware detection (resp. classification).

Website Fingerprinting. The aim is to classify which website (and, at a finer level, which webpage) has been visited by a user via its traffic inspection, among a set of websites that an eavesdropper is monitoring. For mobile networks, this subject is also related to the specific implementation of mobile browsers and data leaked by smartphones (*e.g.*, Android devices track the amount of incoming/outgoing traffic on a per-app basis for data-usage monitoring). Since websites may be targeted for censorship, this subject has a direct impact on the network privacy aspect.

Traffic Identification and Classification. *Traffic Identification (TI)* consists in identifying a specific application (or protocol) among the network traffic, modeled as a binary classification task (*i.e.* application vs. other). Differently, *Traffic Classification (TC)* discriminates several applications (or protocols) among the network traffic and constitutes a multi-class generalization of TI. Besides monitoring goals, TI and TC outcomes are capitalized in enforcing specific policing rules to the targeted application (or class of applications) traffic, such as prioritization, throttling, or blocking. This leads to a finer network management. Also, TI and TC are both tightly-coupled to the privacy aspect, for instance recognition of context-sensitive apps in mobile scenarios. Lastly, both also have security applications, such as detection of unexpected or unauthorized network services that, although not malicious in nature, either expose a wider attack surface or violate policies (*e.g.*, advertisers and analytics). A deeper analysis of TI and TC is the object of the next section, that underlines their main characteristics and requirements, together with the challenges arsing in encrypted and mobile traffic-context.

1.3 An Overview on Traffic Classification

The process of associating (labeling) network traffic with the specific applications or application types generating it is known as *Traffic Classification*. Differently, when the task is recognizing only the traffic generated by one specific application among the others, it is referred to as *Traffic Identification*. TC has been a critical task in network monitoring and management for over a decade and has involved operators, researchers, and developers working in several fields, being, consequently, backed by a wide scientific literature [29, 30, 31, 32]. It is of utmost importance in Internet traffic engi-

neering, along with related methodologies and tools, as they jointly support activities such as network monitoring, security assessment, application identification, accounting, advertising, and service differentiation. For example, ISPs use classification to enforce traffic management policies such as video traffic throttling, VoIP traffic prioritization, content-sensitive pricing, and censorship circumvention [33, 34, 35], whereas Content Delivery Networks (CDNs) use TC to optimize their network for better content delivery and quality-of-service guarantees [36, 37].

In the following, TC timeliness and granularity (§1.3.1), the means to evaluate its performance (§1.3.2), and the approaches proposed for effective TC (§1.3.3) are discussed. Finally, the workflows of TC via ML and DL (§1.3.4) are introduced, showing that they are particularly suitable for working with encrypted and mobile traffic.

1.3.1 Traffic Classification Timeliness and Granularity

The *timeliness* of TC is defined on the basis of the amount of traffic needed before emitting the verdict. In this regard, *"early" TC* [38, 39] can classify the traffic after a few bytes or packets, being particularly deemed for concerns needing a swift action (*e.g.*, security), as opposed to *"late"* (viz. flow-based/post-mortem) *TC* that must wait for traffic object termination and is suitable for concerns such as network management and privacy monitoring.

Additionally, the main factors that determine the *granularity* of TC are the considered traffic object and the traffic classes to which these objects can be ascribed, being indeed the natural *input* and *output* of the traffic classifier, respectively. As expected the finer the granularity, the harder the TC task.

Traffic Object. TC literature has considered different traffic objects. The definition of a specific traffic object determines how raw traffic is segmented into multiple discrete traffic units [30].

The most-common traffic objects are the *flow* and the *bidirectional flow* (shortly *biflow*). The former is defined as the set of all the packets having the same 5-tuple (*i.e.* source IP, source port, destination IP, destination port, and transport-level protocol) taking into account also their directions. Differently, a biflow includes both directions of traffic sharing a given tuple (*i.e.* the source and the destination are interchangeable). The biflow direction is defined according to its first packet: the packet source (destination) is chosen as source (destination) for the whole biflow. Some definitions of flow and biflow include also an additional timeout (*i.e.* idle time or periodic reset) to determine its termination [40].

In addition to flows and biflows, other common traffic objects are: (i) *TCP connections*, identified based on the observation of TCP flags (*i.e.* SYN, ACK, FIN, etc.) or the utilization of TCP state machines; (ii) *services*, including all the packets sharing the same transport-level protocol, source IP, and port; (iii) *bag of flows*, including all the packets sharing the same transport-level protocol, destination IP, and port; (iv) *hosts*, considering both the transmitted and received traffic.

Furthermore, *burst* and *Service Burst (SB)* are traffic objects specifically employed in the mobile-phone identification and mobile-app classification domains. The burst [41] is a sequence of packets having an inter-packet time smaller than a given threshold (named Burst Threshold (BT)), irrespective of their source or destination addresses, as well as of the biflow they belong to. Accordingly, a SB [42, 43] is then a set of packets, within a single burst, that belongs to biflows sharing the same transport protocol, destination IP address, and port number (*i.e.* to the same bag of flows). It is worth noting

that the BT is a key (tunable) parameter in the definition of bursts and SBs.[2]

Finally, when the traffic object is the single *packet* (*i.e.* the classification is performed at packet level), it leads to the finest granularity for a TC problem and virtually represents the hardest setup for the corresponding classification task.

Traffic Class. The traffic class represents the category to which each traffic object can be assigned. Traffic classes mainly depend on the specific domain in which TC is performed. For example, the classes considered in anonymous traffic classification are related to the specific anonymous network taken into account (*e.g.*, Tor Pluggable Transport or I2PSnark) [44], whereas in the case of anomaly/malware detection, the focus is on discriminating between normal/malicious traffic and classifying the different malware types, respectively [17, 21].

Nevertheless, traffic classes of different granularities commonly used in TC can still be identified: (i) *Protocol*, *e.g.*, HTTP, FTP, SMTP, etc.; (ii) *Traffic Type*, *e.g.*, interactive, bulk, video, audio, P2P, etc.; (iii) *Application Category*, *e.g.*, web browser, email, chat, social networking, news, etc.; (iv) *Application*, *e.g.*, Firefox, Outlook, Skype, Facebook, WhatsApp; (v) *Content Type*, *e.g.*, text, binary, picture, etc.; (vi) *Action*, *e.g.*, sending a message, posting a content, registering/logging-in to a service, etc. Notably, these classes can be further inflected depending on the specific domain.

[2]SB notion has been introduced in [42, 43] under the (different) name of flow. However to avoid any ambiguity with the common and established definition of flow, the decomposition used in [42, 43] is herein referred to as SB.

1.3.2 Traffic Classification Evaluation Measures

The evaluation of the performance of traffic classifiers requires a rigorous set of well-defined measures that could be used regardless of the specific TC approach employed. All these measures are obtained comparing the output traffic class with the actual traffic class, commonly referred to as the *ground-truth*.

The most commonly used measure is the *accuracy*, being the fraction of correctly classified instances among the total number of instances. Accuracy is sometimes referred to as *overall accuracy* or when the flow/biflow is the traffic object, as *flow accuracy*. However, some works [45, 46] have also utilized the *byte accuracy*, which is the ratio of the sum of all bytes carried by the correctly classified traffic objects to the sum of all bytes in the traffic considered [32]. This latter measure is more related to the network operational scenario and more robust to the class imbalance problem afflicting ML-based classifiers (cf. §1.3.4). Moreover, the concept of *Top-K accuracy*—recently used in Website Fingerprinting (WF) [47]—is also employed when the classifier outputs not only the traffic class but also the set of prediction probabilities (viz. soft-outputs) for each of the L classes. It defines a correct classification event if the true class is within the top K predicted labels ($K < L$ is a free parameter[3]), allowing to investigate the soft-output behavior of a multi-class classifier.

In addition to accuracy, basic measures are instead defined for a binary-classification problem (*i.e.* discriminating between two traffic classes A and B) or a binary-equivalent one (*i.e.* discriminating between class A and the others, *e.g.* TI). Referring to the latter more general case, four possible measures can be identified:

[3]Of course $K = 1$ coincides with the standard accuracy.

- *True Positive Rate (TPR)*: the percentage of samples of class A *correctly* classified as belonging to class A.

- *True Negative Rate (TNR)*: the percentage of samples of other classes *correctly* classified as *not* belonging to class A.

- *False Positive Rate (FPR)*: the percentage of samples of other classes *incorrectly* classified as belonging to class A.

- *False Negative Rate (FNR)*: the percentage of samples of class A *incorrectly* classified as *not* belonging to class A.

Other measures are borrowed from the ML domain but could be seamlessly applied to all TC approaches. These are also defined on per-class basis:

- *Precision* (prec): the proportion of classifier decisions for a given class which are actually correct.

- *Recall* (rec): the class-conditional accuracy.

It is worth noting that, in the ML context the terms sensitivity (sens) or recall and specificity (spec) are used to refer to TPR and TNR, respectively.

Usually, to account for the effects of multiple measures concisely, composite metrics are considered:

- *F-measure*: the harmonic mean of precision and recall ($\mathrm{F} \triangleq (2 \cdot \mathrm{prec} \cdot \mathrm{rec})/(\mathrm{prec} + \mathrm{rec})$).

- *G-mean*: the squared root of the product of the sensitivity (recall) and specificity ($\mathrm{G} \triangleq \sqrt{\mathrm{sens} \cdot \mathrm{spec}}$).

To leverage these per-class measures for the evaluation of a multi-class traffic classifiers, their *arithmetically averaged* (viz. macro) versions are employed.

Furthermore, to analyze the whole performance "picture" and identify the most frequent misclassification patterns of a traffic classifier, the *confusion matrices* are commonly used. Specifically, they report the predicted classes on the rows and the actual classes on the columns. Thus, all the predictions not located on the diagonal of the matrix are misclassifications, whereas a higher concentration toward the diagonal implies better performance.

Finally, to provide a complete performance view, classifiers can be also tested when they are enriched with a "reject option"[4], namely the classification is performed only if the highest class prediction probability exceeds a threshold γ and "unsure" classifications are then *censored*. This evaluation is extremely interesting in the mobile traffic context [43], since mobile apps typically send multiple flows when used, there remains high chance to identify them from their more distinctive flows, without the need to classify all the instances (*i.e.* the classifier does not reach a verdict when the highest class prediction probability is below γ). Hence, tuning γ can be effective to improve classification performance while incurring negligible drawback, *i.e.* a decreased ratio of classified flows.

1.3.3 Approaches to Traffic Classification

The evolution of methods employed to perform TC has been dictated by the evolution of Internet traffic and the resulting increasing complexity to perform the classification task.

[4]In this case, the classifier must output also the class-prediction probabilities.

Port-based. The first and simplest method for TC leverages the TCP/UDP port-number information, mapping applications to ports. Indeed historically, many applications have used "well-known" port numbers to offer their services to remote hosts (*e.g.*, 80 for HTTP, 443 for HTTPS, 25 and 110 for email, 53 for DNS, etc.). Thus, a port-based classifier is extremely simple and fast, since it can operate by only accessing the transport-level header of the packets, without needing complex computations. However, the changing traffic nature has made this approach less and less accurate. Indeed, many applications (i) do not have IANA-registered ports and use custom or randomly-assigned ports, (ii) hide their traffic behind HTTP(S) (*i.e.* toward port 80(443)) to pass through firewalls and avoid filtering, (iii) use network address translation and dynamic port allocation [30]. Therefore, this approach is useful only when the speed and simplicity of classification are more urgent than accuracy.

Payload-based. To raise TC accuracy, methods based on the full payload inspection—also known as Deep Packet Inspection (DPI)—have arisen. Specifically, they implement pattern-matching techniques that compare the content of packets with predefined byte-sequences or signatures, characteristic of a specific traffic type or application [48, 49, 50]. Although DPI is more reliable than simply relying on port information, it is extremely computational-expensive and needs accurate and up-to-date signature bases, being thus unfeasible to keep the pace of high bandwidth traffic and perform the concurrent classification of a large number of traffic objects. Moreover, the payload inspection clashes with privacy policies and legislation [51, 52] that might impede direct analysis of application-layer content. Despite the attempts done to mitigate these issues combining lightweight DPI with port-based method [53], the increasing adoption of encrypted protocols

(TLS) [54, 55] makes this type of classification even more challenging, defeating also established approaches.

Behavior-based. Some works try to complement payload-based classification with different heuristics based on host or application behaviors. For example, they leverage constraints useful to simplify the onerous payload inspection of each traffic flow, as classifying one flow through DPI and then associating the same application to the other unknown flows having the same destination IP and port number [56, 57]. Another example of behavioral classification is based on observing and identifying patterns of host behavior at the transport layer using only IP address and port information and the role of a host in the network (*i.e.* provider or consumer of a service) [45]. The main drawback of these methods is the need to gather information from several flows generated by different hosts or applications before they can emit a verdict, being thus infeasible for real-time operational networks.

Statistical Traffic Classification. To overcome the limitations of previous methods, in the last years an increasing number of approaches relying on statistical properties of traffic objects have been proposed. In other words, they try to exploit the peculiar statistical characteristics of traffic generated by a certain application (*e.g.*, the distribution of packet lengths or inter-arrival times) to infer the application itself [32]. In this context, traffic classifiers based on *Machine* and *Deep Learning* have proved to be promising, being able to cope also with *encrypted traffic* that hinders the utilization of payload-based methods instead.

1.3.4 Traffic Classification using Machine and Deep Learning

In 1968, Donald Michie [58] stated:

> *"It would be useful if computers could learn from experience and thus automatically improve the efficiency of their own programs during execution. A simple but effective rote-learning facility can be provided within the framework of a suitable programming language."*

This intuition has laid the groundwork for the definition of ML that Shi [59] in 1992 defined as:

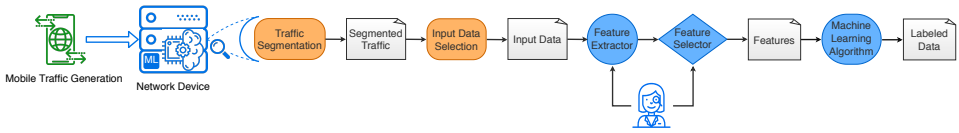
> *"Machine learning is the study of making machines acquire new knowledge, new skills, and reorganise existing knowledge."*

In addition to network traffic analysis (cf. §1.2), a wide range of applications have exploited the ML paradigm. These includes natural-language processing, image recognition, video surveillance, financial trading, fraud detection, search engines, virtual personal assistants, and many others.

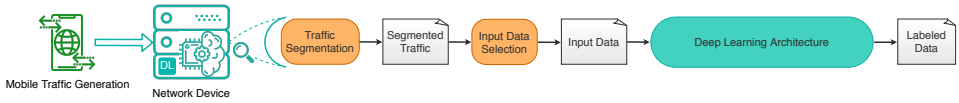
As mentioned just before, the researchers have shown huge interest in the application of ML also to TC to go beyond the limits of other approaches. Figure 1.2a shows the workflow of an ML-based (mobile) traffic classifier, highlighting its key steps.[5]

Firstly, the traffic is segmented into relevant traffic objects being the classification samples (cf. §1.3.1). A successive step of paramount importance for ML-based TC is defining and extracting a set of *features* (usually referred as *feature engineering* and *extraction*, respectively) from *input data*

[5]It is worth noting that the main steps of the reported workflow are independent from the specific nature of network traffic (*i.e.* mobile vs. fixed).



(a) Traditional Machine Learning Flow.



(b) Deep Learning Flow

Figure 1.2: Traditional ML flow (a) vs. DL flow (b). For this latter the feature extraction and selection are delegated to the DL architecture, without needing human intervention.

that are able to synthesize peculiar characteristics of traffic objects useful to identify and discriminate between the traffic classes. Indeed, the successful use of ML classifiers relies on obtaining handcrafted (domain-expert driven) features, which in TC context usually correspond to information extracted from the sequence of packets [60, 43] or message sizes [61, 62]. For example, ML techniques may be applied either directly on the whole sequence (such as in [43, 61, 63]) or based on statistics/histograms extracted from it (such as in [43, 63, 64, 65]). It is worth noting that the above statistical techniques can be also combined with port-based algorithms (in scenarios where port-info can be considered reliable) to develop hybrid approaches, such as [66]. The extracted feature set can be further refined (not necessarily) to remove unnecessary or redundant features that have no or detrimental effect on classification performance. This process is known as *feature selection*. Indeed, having the smallest necessary set of features needed to attain certain accuracy is a key advantage for an ML algorithm in an operational scenario (*i.e.* the lesser the features, the simpler and faster the classifier). Feature selection can be performed only on the basis of the traffic data character-

istics (*e.g.*, removing features based on their variance or using univariate statistical tests) or can be dependent from the specific algorithm employed for TC (*e.g.*, removing features recursively based on the performance the classifier achieves).

Then, the features are fed to the *Machine Learning algorithm* for the *training* (also known as learning). The training phase strictly depends on the specific algorithm considered. However, three main families can be identified on the basis of the previous knowledge about the traffic classes each sample (viz. traffic object) of the training (data)set belongs:

- *Supervised Learning* requires prior knowledge on the classes (viz. labels) of the traffic objects within the training set. The classifier learns the association rules between the features and traffic classes, that is the input-output patterns. It is worth noting that the labeling (*i.e.* the ground-truth quality) deeply influences the training process and consequently the classification performance.

- *Unsupervised Learning* does not require prior knowledge about the traffic classes in the training set (*i.e.* it works with unlabeled data). The classifier learns patterns in the input data and groups samples based on shared attributes or commonalities (*i.e.* clustering). Unlabeled data is less expensive and simpler to collect but at the end of the training phase, a domain expert must label the resulting clusters to map unknown samples to traffic classes.

- *Semi-supervised Learning* requires a small number of labeled samples and a large number of unlabeled ones belonging to the same set of traffic classes. The classifier can both learn patterns in input data and simultaneously associate the clusters to labels, combining the advan-

tages of supervised and unsupervised approaches, but without guaranteeing better classification performance [67].

The trained model is *tested* classifying unseen samples (*i.e.* test set). Having prior knowledge on the classes of the test samples (viz. ground-truth), one can evaluate the performance of the classifier by comparing its output class or the resulting (labeled) cluster with the ground-truth, in the case of supervised or unsupervised learning, respectively.

As underlined earlier, the feature extraction and (if carried out) selection steps are fundamental to design an accurate and up-to-date traffic classifier. Moreover, constant human-expert intervention is required to accomplish this goal. Sadly, such process is time-consuming, unsuited to automation, and it is becoming rapidly outdated when compared to the evolution and mix of network traffic, being a constantly *moving target*. This is even exacerbated in the case of mobile traffic classifiers realized with "traditional" ML approaches.

An advancement of ML-based TC is exploiting DL which allows training classifiers directly from input data by *automatically learning* structured (and complex) feature representations instead of relying on manually-designed features. The terms "deep" refers to the usage of multiple transformation steps to create these features, which is reflected in computations performed by a deep neural network made of many "hidden" layers placed between the input layer (passing input data to the first hidden layer) and the output layer (producing the output variables).

Figure 1.2b depicts the workflow of a DL-based (mobile) traffic classifier, reporting its main steps. Comparing DL with traditional ML workflow, it can be noticed that the feature extraction and selection building blocks are delegated to the DL architecture, without needing domain-expert inter-

vention. Nevertheless, DL classifiers require also unbiased and informative *input data* that derive from domain-expertise, thus reducing, but not eliminating the human intervention.

As stated above, the structure of the *Deep Learning architecture* is a neural network whose computation unit is the neuron (or node). Each node receives (weighted) inputs from other nodes (or from the external if it belongs to the input layer) and produces an output applying a non-linear function f called *Activation Function* (*e.g.*. Sigmoid, Rectified Linear Unit, hyperbolic tangent, etc.) to the weighted sum of its inputs [68].

Similarly to ML classifiers, DL architectures can be gathered based on their learning procedures into supervised, unsupervised, and semi-supervised. However, DL architectures are trained in an *iterative* fashion leveraging the *stochastic gradient descent* (first-order) optimization algorithm for finding the minimum of a cost (or loss) function. Specifically, it calculates an estimate of the gradient from a random subset of the training data instead of considering the entire dataset. Three hyperparameters highly impact this learning procedure:

- *Training Epochs* determine the times the whole training set is passed forward and backward through the deep neural network (*i.e.* the times the gradient descent works on the complete training set).
- *Batch Size* represents the total number of training samples present in a single batch. Indeed, for computational constraints, the training set is further divided into smaller parts (or number of batches). The number of batches represents the iterations needed to complete one epoch.
- *Learning Rate* controls to what extent the model must change in response to the estimated error in each step. Since the learning rate

determines how fast the model converges, the larger the learning rate, the fewer the training epochs required. However, a learning rate excessively large can cause the model to converge to a suboptimal solution.

Final Remarks. TC comes with its own challenges and requirements that are even exacerbated in the encrypted and mobile traffic-context, which hinder the achievement of satisfactory performance with standard methods. Indeed, the increasing number of (mobile) applications communicating with online services using HTTPS and the consequent adoption of standard encrypted protocols (TLS) [69], and the proliferation of privacy-preserving tools (*e.g.*, anonymity tools) [70]—additionally hiding the source, the destination, and the nature of the communication—make the classification more challenging, defeating established approaches (*i.e.* port- and payload-based) and nourishing the adoption of advanced ML- and DL-based classifiers.

Also, achieving targeted TC performance in the mobile context is further undermined by the large number of apps to discriminate from and the adoption of a successful multi-platform framework-based development and distribution model [71], implying: (i) the embedding of common (third–party) network services to implement app features; (ii) the quick proliferation of (similar) apps to discriminate from; (iii) a fast-paced update cycle of apps, development frameworks, and operating systems. For TC all these characteristics impair app-fingerprint collection, definition, and update, also possibly reducing the number of training samples available per app, due to limited time between updates.

1.4 Contribution and Background

Although earlier results have been published on TC in this domain, the encrypted traffic of mobile apps represents a moving target for classifiers due to its dynamic evolution and mix. Therefore, classifying this traffic constitutes an open and evolving research field. In the following, Sec. 1.4.1 provides a summary of the main contributions of this Thesis, highlighting also the set of *Research Questions (RQs)* it aims to answer, then Sec. 1.4.2 sets these contributions in the big picture of most related literature, leaving to pertinent chapters its detailed analysis.

1.4.1 Summary of Contributions

This Thesis presents a set of novel methodologies for mobile and encrypted TC, proposing various advancements with respect to the state-of-the art, taking into account the peculiarities and challenges of TC performed in this domain. In detail, different possibilities to improve current solutions are identified, implemented, analyzed and evaluated separately to clearly highlight in which way and to which extent each of these might be beneficial but also to discuss their challenges, possible limitations, and open issues. Nevertheless, it is necessary to stress that these proposals could–and indeed should–be envisioned non only as alternatives, but as components of a comprehensive composite framework explicitly devised for mobile and encrypted TC and able to deal with its unique traits.

As mentioned in Sec. 1.3, starting from the standpoint that current classification approaches fail to face issues in encrypted and mobile traffic, with also traditional ML classifiers revealing some weak spots in this dynamic context, the present dissertation shows how these latter might be improved, proposing two advanced "structural" design choices against basic ML-based

TC.

The first advancement proposed is a *Multi-Classification (MC)* system which intelligently combines the decisions of traditional ML classifiers, based on both hard and soft approaches (*i.e.* taking into account only the classifiers' hard decision or also their class-prediction probabilities, respectively). The implemented MC architecture leverages as (base) building blocks the state-of-the-art classifiers specifically devised for mobile- and encrypted-TC. Indeed, it can potentially overcome the deficiencies of each single classifier and provide improved performance with respect to any of these, also allowing for modularity of classifiers' selection in the pool. By proposing and evaluating such MC system, this contribution aims to answer the following *RQ_1: to what extent is it possible to improve the classification performance of mobile apps taking the best from each state-of-the-art ML classifier via advanced combining techniques?*

The MC scheme entails an "horizontal" enhancement in the classifier structure, being still however a flat approach to TC. Indeed, a "vertical" enhancement can be further envisioned resorting to a structure of (potentially different) classifiers (viz. nodes) arranged in a tree fashion, each specialized in labeling a subset of (potentially more and more finer) classes (cf. §1.3.1). This hierarchical framework exploits the "divide-et-impera" principle, enabling the partition of the workload among several classifiers and grants scalability and modularity. Specifically, the obtained *Hierarchical Classification (HC)* structure permits fine-grained (*i.e.* per-node) design, tuning and evaluation, potentially leading to classification performance gains, and takes advantage of by-design benefits, as focused re-training and incremental update of specific nodes or distributed deployment of TC tasks. Expressly, the present investigation focuses on the *RQ_2: is it possible to capitalize the inherent hierarchical class-structure of (encrypted) traffic to achieve various*

advantages in TC at increasingly finer granularity?

It is worth noting that multi- and hierarchical-classification could be seamlessly blended into a framework in which each node is not a simple ML classifier but a more advanced MC architecture, with the further ability of fine-tuning the specific MC technique employed in each node of the hierarchy based on the specific TC task to carry out.

In addition to structural enhancements, one can also act to improve the elemental classifiers. Sec. 1.3.4 has shown the main differences in ML and DL workflows for TC, underlining the suitability of this latter to the dynamic and challenging mobile context as opposed to traditional ML-based classifiers. Despite being extremely promising, the naïve adoption of DL to mobile and encrypted TC might entail misleading design choices and lead to biased conclusions, due to the peculiar (and tricky) nature of network traffic data. Therefore, this Thesis proposes the realization of mobile traffic classifiers able to operate with encrypted traffic via DL, developing a systematic framework for the design of novel DL-based TC architectures. Leveraging this framework, existing solutions declined in the mobile scenario are firstly compared, underlining the deficiencies of current DL-based traffic classifiers and providing a fine-level performance evaluation workbench as groundwork for their accurate design. Then, a novel *multi-modal DL-based mobile traffic classification (MIMETIC)* approach is devised and evaluated. MIMETIC is able to exploit the intrinsic multi-modal nature of traffic data and can capitalize the different views (viz. modalities) of the same traffic object (*e.g.*, raw payload or header fields) with an effective improvement over both single-modal DL-based traffic classifiers and their combination (viz. MC). In this regard, it is worth noting that MIMETIC, capturing both intra- and inter-modalities dependence, constitutes a generalization of the MC framework, although being also usable in conjunction with this lat-

ter that would accomplish the fusion of the base multi-modal classifiers' decisions. Therefore, the research question addressed here is *RQ_3: how can mobile and encrypted TC benefit from the domain-aware application of DL to capitalize the heterogeneous traffic nature and avoid biased outcomes?*

Another possible limitation of DL-based TC is the generation of learning networks with very dense and complex structure, whose training might be excessively slow and computational demanding with respect to the timeliness and computational constraints of network domain, being even more urgent in the mobile scenario (cf. §1.1). To solve this issue the present manuscript suggests the integration of the *Big Data (BD)* framework with DL-based TC architectures. BD solutions provide processing frameworks that can parallelize the classification task by splitting the network data and distributing it across different workers cooperating under the coordination of a single master. However, they should be carefully investigated in the case of non-naturally-parallelizable tasks, like the optimization of DL training procedure. With this aim, the usage of DL-based TC strategies as supported by BD frameworks is also evaluated considering classification performance, training completion time, and (cloud-implementation) costs and highlighting relevant non-trivial trade-offs. Hence, the pertinent research question to which the latter contribution answers is *RQ_4: which are the implications on TC when adopting BD solutions to deal with the demanding training phase of DL-based traffic classifiers?*

The data-driven TC approaches presented herein require a reliably-labeled dataset (viz. with a reliable ground-truth), possibly human-generated, to ensure proper design, realization, and evaluation. Therefore, to provide the foundation for the set of proposed TC methodologies, the Thesis presents, as the first contribution, the design and implementation of an architecture for *mobile-app traffic capture and ground-truth creation*

(MIRAGE) expressly conceived to support mobile traffic analysis tasks. The outcome of this architecture is the human-generated *MIRAGE-2019* dataset that has been publicly released with the goal of advancing the state-of-the-art in mobile-app traffic analysis, fostering its replicability and reproducibility. Indeed, the versatility of the MIRAGE-2019 dataset is proven by performing example tasks related to mobile traffic characterization and modeling and employing (part of) it as common benchmark for the experimental evaluation of devised approaches.

1.4.2 Background Overview

Table 1.2 reports the studies more closely related to this Thesis tackling mobile TC—regardless of the specific technique—or standard TC employing at least one of the techniques exploited in the set of proposed methodologies.[6] It can be noticed that few works have taken into account the traffic generated by mobile apps, leveraging either payload-based approaches under the assumption of cleartext traffic [78, 89] or traditional ML-based ones in the case of encrypted traffic [41, 80, 84, 86, 60, 43, 97]. In this regard, the vast majority of non-mobile studies proposed approaches able to deal with encrypted traffic, with the only exception of the works in [81] and [82, 83], leveraging a set of DL classifiers based on plain-text (HTTP) data and an MC/HC system comprising payload-based method, respectively.

Taking into account the techniques employed, earlier works mainly focused on MC frameworks, although none of them have analyzed mobile traffic. Similar considerations could be inferred for HC, that, however, has also had recent applications in combination with DL [96]. Indeed, this latter has experienced enormous interest in the last few years due to its intrin-

[6] A corpus of works not shown in Tab. 1.2 dealt with encrypted TC by means of standard ML-based approaches.

Table 1.2: Mobile and encrypted TC works and comparison with the set of proposed methodologies.
Traffic: Mobile Traffic (MT), Encrypted Traffic (ET).
Technique: Multi-classification (MC), Hierarchical Classification (HC), Deep Learning (DL), Multi-modal (MM), Big Data (BD).
Dataset: Human Dataset (HD), Open Dataset (OD).

		Traffic		**Technique**					**Dataset**	
Paper	**Year**	**MT**	**ET**	**MC**	**HC**	**DL**	**MM**	**BD**	**HD**	**OD**
Szabo *et al.* [72]	2007	□	■	♦	◊	◊	◊	◊	●	○
He *et al.* [73]	2008	□	■	♦	◊	◊	◊	◊	●	○
Callado *et al.* [74]	2010	□	■	♦	◊	◊	◊	◊	●	○
Yu *et al.* [75]	2010	□	■	◊	♦	◊	◊	◊	●	○
Dainotti *et al.* [76]	2011	□	■	♦	◊	◊	◊	◊	●	○
Mellia *et al.* [77]	2012	□	■	◊	♦	◊	◊	◊	●	○
Dai *et al.* [78]	2013	■	□	◊	◊	◊	◊	◊	○	○
Stöber *et al.* [41]	2013	■	■	◊	◊	◊	◊	◊	●	○
De Donato *et al.* [40]	2014	□	■	♦	◊	◊	◊	◊	●	○
D'Alessandro *et al.* [79]	2015	□	■	◊	◊	◊	◊	♦	○	◐
Wang *et al.* [80]	2015	■	■	◊	◊	◊	◊	◊	◐	○
Wang [81]	2015	□	□	◊	◊	♦	◊	◊	●	○
Yoon *et al.* [82, 83]	2015	□	□	♦	♦	◊	◊	◊	◐	○
Alan and Kaur [84]	2016	■	■	◊	◊	◊	◊	◊	○	○
Shbair *et al.* [85]	2016	□	■	◊	♦	◊	◊	◊	●	○
Conti *et al.* [86]	2016	■	■	◊	◊	◊	◊	◊	○	○
Saltaformaggio *et al.* [60]	2016	■	■	◊	◊	◊	◊	◊	●	○
Yuan and Wang [87]	2016	□	■	◊	◊	◊	◊	♦	●	◐
Dong *et al.* [88]	2017	□	■	◊	♦	◊	◊	◊	●	○
Li *et al.* [89]	2017	■	□	◊	◊	♦	◊	◊	●	○
Chen *et al.* [90]	2017	□	■	◊	◊	♦	◊	◊	◐	○
Lopez-Martin *et al.* [91]	2017	□	■	◊	◊	♦	◊	◊	●	○
Lotfollahi *et al.* [92]	2017	□	■	◊	◊	♦	◊	◊	●	◐
Vu *et al.* [93]	2017	□	■	◊	◊	♦	◊	◊	●	◐
Wang *et al.* [21]	2017	□	■	◊	◊	♦	◊	◊	●	◐
Wang *et al.* [94]	2017	□	■	◊	◊	♦	◊	◊	●	●
Ke *et al.* [95]	2017	□	■	◊	◊	◊	◊	♦	●	◐
Taylor *et al.* [42, 43]	2018	■	■	◊	◊	◊	◊	◊	○	○
Chen *et al.* [96]	2018	□	■	◊	♦	♦	◊	◊	●	◐
Le *et al.* [97]	2018	■	■	◊	◊	◊	◊	♦	●	○
Huang *et al.* [23]	2018	□	■	◊	◊	♦	◊	◊	●	◐
Shi *et al.* [98]	2018	□	■	◊	◊	♦	◊	◊	●	◐
Zhang *et al.* [99]	2018	□	■	◊	◊	♦	◊	◊	●	◐
Wang *et al.* [100]	2018	□	■	◊	◊	♦	◊	◊	●	◐
Li *et al.* [101]	2018	□	■	◊	◊	♦	◊	◊	●	○
Liu *et al.* [102]	2019	□	■	◊	◊	♦	◊	◊	●	○
Sun *et al.* [103]	2019	□	■	◊	◊	♦	◊	◊	●	◐
Zeng *et al.* [104]	2019	□	■	◊	◊	♦	◊	◊	●	◐
This thesis	*2020*	■	■	♦	♦	♦	♦	♦	●	●

sic ability to work with encrypted traffic and significantly reduce domain-expert (viz. human) intervention in the design of classifiers (cf. §1.3.4). Nevertheless, up to our knowledge, no previous studies have systematically applied DL for both mobile and encrypted TC. Additionally, while single-modal DL has been widely employed, multi-modal approaches have not been considered yet, despite being able to better capitalize heterogeneity of network data. Similarly, even though BD represents a fitting accelerator for (computationally-demanding) training and test phase of DL-based traffic classifiers, it has not been utilized in this context, but only along with ML-based classifiers and marginally in the mobile domain [97].

Finally, Tab. 1.2 reports also the kind of traffic data exploited for the design and evaluation of TC approaches reviewed, highlighting if network data is human-generated and if it has been released as an open dataset. Notably, the works tackling mobile TC are more prone to exploit automatically-generated (viz. non-human) datasets due to the usage of automated UI exerciser "monkeys" that automatically test apps using randomized input [78, 84, 86, 42, 43]. Unfortunately, monkeys may generate results that are not representative of normal user interaction with the app [105]. Additionally, an half-circle in the "Human Dataset" column indicates that the corresponding work does not clearly state if the traffic is human-generated [81, 90] or it does not perform the experimental evaluation of its proposal at all [82, 83]. Moreover, only Wang *et al.* [21] have publicly released their dataset (being however tailored for malware classification), whereas other papers employed either private (blank circle) or publicly-available (half-circle) datasets (*i.e.* not generated by the authors themselves). Interestingly, the majority of these studies have employed only two public human-generated datasets, namely the "Moore" [106] and "ISCX VPN-nonVPN" [107] datasets that do not include mobile app traffic, highlighting the lack of an up-to-date human-

generated dataset with associated ground-truth for mobile and encrypted TC.

1.5 Organization of the Thesis

The rest of the Thesis is organized as follows. Chapter 2 describes the MIRAGE architecture for mobile-app traffic capture and ground-truth creation and the MIRAGE-2019 dataset obtained by means of MIRAGE and (partially) employed for the experimental evaluation of devised TC methodologies. Chapter 3 details the proposed Multi-Classification System and shows how it is able to achieve better performance than each single base classifier, enjoying also the modularity given by readily plugged-in/out of its components. Chapter 4 introduces the Hierarchical Classification approach and demonstrates how it can deal with the challenging task of encrypted TC of anonymity tools; the proposed hierarchical approach is compared with a pool of flat counterparts. The design of mobile traffic classifiers via DL is presented in Chapter 5; it defines a systematic framework exploited for the tuning and comparison of existing single-modal DL-based approaches and the design of the novel MIMETIC multi-modal architecture that is proved to outperform the best single-modal baselines. The integration of the BD accelerator for the deployment of DL-based traffic classifiers is discussed in Chapter 6; it is shown how BD paradigm cannot be transparently applied in this scenario underling the need for a careful evaluation shedding light on deployment trade-offs. Finally, conclusions are drawn in Chapter 7.

Chapter 2

The MIRAGE Architecture

In this chapter, we introduce and describe *MIRAGE (Mobile-app traffic capture and ground-truth creation)*, a reproducible architecture for the capture of mobile-app traffic and the creation of the related Ground Truth (GT). The outcome of this system is *MIRAGE-2019*, a labeled human-generated dataset suitable for mobile (encrypted) traffic analysis that we have publicly released[1] with the goal of advancing the state-of-the-art in this field.

In next sections, we provide the motivations that led us to design and implement MIRAGE and review the current publicly-available datasets for encrypted (and possibly mobile) traffic analysis (§2.1). Then, we give the details of the whole MIRAGE architecture along with traffic-capture and GT-building phases (§2.2) and describe the release format of the MIRAGE-2019 dataset (§2.3). Finally, we perform a characterization of MIRAGE-2019 proving that it can be capitalized for different tasks related to mobile traffic analysis (§2.4), encompassing its utilization (in conjunction with other datasets) as a benchmark for the experimental evaluation of proposed

[1] MIRAGE-2019 is publicly released at http://traffic.comics.unina.it/mirage.

TC methodologies (§2.5).

2.1 The Need for the MIRAGE Solution

As in almost all experimental-research fields, *replicability* and *reproducibility* are critical concerns in achieving significant and grounded progress [108]. Accordingly, a strong push for them is currently observed, with research artifacts being also evaluated in the standard peer-review process [109]. In fact, provisioning datasets as well as carefully documenting workflows for obtaining them, are critical to foster replicability and reproducibility, respectively, with both fueling research dissemination. Indeed, by leveraging wisely sourced and constructed datasets that are available to the research community, research outcomes become (*i*) *easier to reproduce*, (*ii*) *comparable* against other studies, and (*iii*) *generalizable* to other data/systems not studied yet.

These possibilities have been sorely missing for a long time [30] in the field of network traffic analysis. In spite of this huge interest, the availability of data for performing research within this field has remained quite limited. Many appealing research solutions have been (and are) mostly validated on private datasets, thus precluding repeatability and safe advances on the topic. Still, it is shared opinion that this aspect contributes to slow down and limit the understanding of the tackled problems, since it constitutes a severe drawback in both design and experimental validation phases.

Even worse, in recent years new challenges have arisen. As described in Chapter 1, the smartphones have become the main medium of communication and also due to their software distribution systems (*i.e.* the apps marketplaces), fostering one-click installation and quick-paced automatic updates, the overall network traffic has reached huge volumes and started

evolving at an unprecedented pace. In this scenario, privacy concerns further limit the collection and publishing of raw mobile traffic traces. These challenges call even more for the availability of mobile-traffic datasets, considering that even when they are made available to the community, the common presence of bot-generated traffic or experiments run in controlled environments (cf. §1.4.2) limit the validity of the proposed analysis or design solutions.

In this context, MIRAGE represents a *reproducible* solution for human-generating new mobile-app traffic datasets and automatically creating the related high accurate GT. Moreover, to foster the *replicability* of traffic analysis and its extension to multiple use cases (including benchmarking of devised methodologies for TC), we have also collected and released the MIRAGE-2019 dataset.

2.1.1 Encrypted Network Traffic Datasets

To underline the urgency of the MIRAGE solution, in the following, we review the most related public traffic datasets released to date in the last years (2014-19), highlighting their main characteristics as well as main shortcomings and limitations, and categorize them according to the taxonomy defined in Tab. 2.1. We point out that we have considered only the datasets collecting encrypted network traffic tailored for TC. We have not taken into account the datasets collected for other purposes, as those related to network anomaly detection or malware/attack classification [21], due to their peculiar focus on network security. Moreover, to reflect only the recent trend toward the growing adoption of encrypted protocols, we have not included "older" datasets (*e.g.*, the widely used Moore dataset [106] released in 2005) since they, although valuable, may be no longer capable to reflect the characteristics of current network traffic.

Table 2.1: Summary of previous datasets on encrypted traffic analysis.

Reference	Dataset	Capture Span	📱	🧍	TO	Raw	Pkt	Stats	Meta	Task	Diversity	R
[107]	ISCXVPN2016	Mar. '15 - Jun. '15	○	●	B	✔		✔		TI&TC	7 TTs	◐
[110]	ISCXTor2016	Lug. '15 - Feb. '16	○	●	BF	✔		✔		TI&TC	8 TTs / 18 apps	◐
[44]	Anon17	'14 - '17	○	●	F		✔	✔	✔	TI&TC	3 ATs / 8 TTs / 21 apps	◐
[111]	QUIC	Mar. '18	○	○	BF		✔			TI&TC	5 QUIC services	◐
[112]	MTD	Oct. '16 - Mar. '17	●	●	BF			✔	✔	TI&TC	12 Apps / 10 DEVs / 10 EXPs	◐
[113]	UNSWIoT	Oct. '16 - Apr. '17	◐	●	BF	✔				DevID	28 DEVs	●
[114]	NTD Reddit	Apr. '18 - May '18	○	○	W	✔				WebAN	5 BWs	◐
[115]	Video Streams	Aug. '15 - May '16	○	○	F	✔				VidID	2.1k VTLs	●
[116]	YouTube Video	Sep. '17 - Feb. '18	●	○	P		✔		✔	VT-QoE	3 VTLs / 374 h	●
[117]	Netflix UE	Oct. '18 - Feb. '19	○	○	F			✔		VT-QoE	10 LOCs / 2.6k VTLs	◐
This Thesis	*MIRAGE-2019*	May '17 - May '19	●	●	BF		✔	✔	✔	TI&TC	40 apps / 3 DEVs / 280+ EXPs	●

Traffic Nature: 📱 = Mobile, 🧍 = Human-generated. **Traffic Object (TO)**: `BF` = Biflow, `F` = Flow, `P` = Packet, `W` = Webpage. **Released Data**: `Raw` = PCAP files, `Pkt` = Packet-level data, `Stats` = TO statistics, `Meta` = Metadata. **Task**: `DevID` = Device Identification, `TI&TC` = Traffic Identification and Classification, `VidID` = Video Identification, `VT-QoE` = Quality-of-Experience in Video Traffic, `WebAN` = Website Analysis. **Diversity**: `AT` = Anonymity Tool, `BW` = Browser, `DEV` = Device, `EXP` = Experimenter, `LOC` = Location, `TT` = Traffic Type, `VTL` = Video Title. **Reproducibility (R)**.

In detail, we categorize each dataset based on whether (*a*) it focuses on the mobile scenario, (*b*) it is generated by real human experimenters (as opposed to bots or scripts), and (*c*) the description of the capture system employed for generating the traffic makes it partially/completely reproducible. As a complementary information, we also provide the capture span for each dataset, either explicitly specified in the corresponding paper or obtained by direct inspection of the artifacts. Additionally, we surface (*i*) the traffic object considered (cf. §1.3.1), (*ii*) the type(s) of released data, and (*iii*) the diversity of the collection space. Specifically, with respect to point (*ii*), we classify the form of released data in *Raw* if PCAP files are available, *Pkt* if fields from each packet are available, *Stats* if summarizing statistics for each traffic object are available, and *Meta* if complementary metadata are available. Differently, referring to point (*iii*), we provide the number of different services/applications or types of objects considered. Finally, an explicit mention to the main intended task approached is provided.

Firstly, referring to the *capture span*, we observe that the experimental campaigns last from months to years, with longer ones typically associated with human interaction (see later discussion). A similar rationale applies to MIRAGE-2019, collected in the last two years by human users and expected to reflect better the current nature of mobile traffic.

Differently, focusing on network traffic generated exclusively by *mobile* (handheld) devices, it is apparent that only the datasets MTD [112] and our MIRAGE-2019 have captured this type of traffic. The only exceptions are represented by UNSWIoT [113] in which some background traffic is generated from mobile devices and YouTube Video [116] where streaming video was analyzed on smartphones.

As anticipated, not all the considered datasets have been generated by *human users*. Indeed, we point out that the above feature may be crucial

when analyzing the traffic generated by complex interaction patterns from the users, as in the case of anonymity tools [107, 44] and mobile apps [112] with automated tools non reflecting completely the above complex behavior. For example, in [111, 114, 115, 117] the authors have employed Selenium [118] for automating web browsing.

Referring to *traffic object* segmentation, most of the works consider either flows [44] or biflows as the relevant traffic analysis unit, with the sole exception of [114] using webpages as the significant object of analysis.

Referring to the main *task* approached, most of the datasets have been collected with the aim of performing and evaluating TI and TC, with specific focus on anonymity tools to assess their degree of anonymity [110, 44], specific traffic services (*e.g.*, Google QUIC protocol) [111], Virtual Private Networks (VPNs) [107], or mobile apps' classification [112]. Differently, other datasets are specifically focusing on (encrypted) video traffic analysis, with either considering title fingerprinting [115] or Quality-of-Experience (QoE) prediction [116, 117]. Finally, some works focus on identifying specific devices generating traffic, such as Internet of Things (IoT) devices [113], and others delve into website analysis [114].

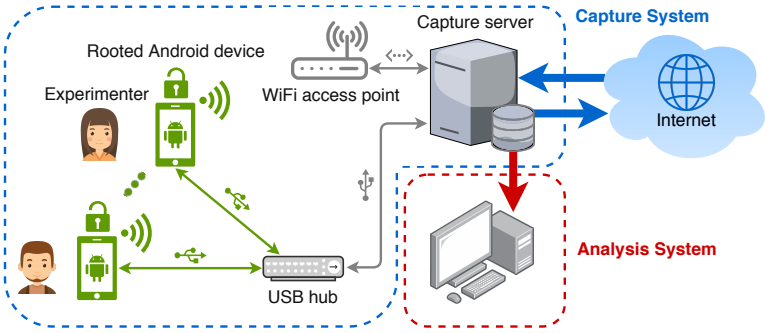
Referring to *diversity*, reviewed datasets provide information with different degrees of variety, also according to the goal and the scope of the work they are proposed with. Overall, different applications/services/-contents are considered, as well as multiple experimenters (in the case of human-generated traffic), capture devices, and locations. For instance, ISCXTor2016 [110] contains 8 different traffic traffic types (browsing, audio, etc.) corresponding to 18 applications which are run under two different scenarios, one to detect Tor traffic flows and the other to detect the application type. Differently, Anon17 [44] contains info about the traffic types and applications running on Tor, I2P, and JonDonym and it is provided in the

form of three-level labels for each flow (cf. §2.5.2). However, each dataset focuses on a limited set of the mentioned aspects in line with the nature of the problem to investigate. In fact, none of those surveyed covers all these aspects at best. At most, 21 applications [44], 28 devices [113], and 10 experimenters [112] are considered. MIRAGE-2019 takes into consideration the traffic generated by 280+ experimenters using 40 mobile applications via 3 devices.

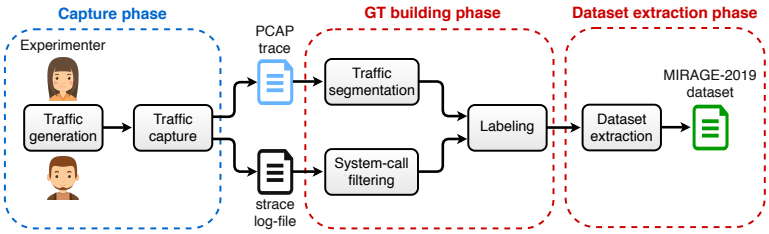
Finally, referring to *reproducibility*, it is apparent that not in all the cases the details, the setup, and the procedures required to reproduce the same experimental environment for the traffic capture have been reported. Nonetheless, in some cases, the authors have also provided a detailed description of the whole experimental environment, see *e.g.* [113, 115, 116].

2.2 The MIRAGE Architecture

At a high level, the MIRAGE system architecture consists of two main components: the *Capture System* and the *Analysis System*. Figure 2.1a shows the architecture of the Capture System. The *capture server* is a workstation equipped with an IEEE 802.11g *access point*, which provides connectivity to the *mobile devices* that generate the traffic when human *experimenters* operate the apps. A wired connection brings the access of the capture server to the public Internet, performing Network Address Translation (NAT). Notably, the upstream connections to the public Internet do not constitute a bottleneck in terms of bandwidth, thus not impacting the properties of the traffic stream flowing through the access WLAN. Each mobile device is also physically attached to the capture server through the *USB hub*; this allows leveraging the Android Debug Bridge (ADB) to send commands from the capture server to the attached devices and receive responses on an off-band



(a) MIRAGE architecture.



(b) MIRAGE-2019 dataset building.

Figure 2.1: The MIRAGE architecture: (a) diagram of main components and (b) workflow of dataset building.

channel. Such procedure requires the devices to be *rooted* in order to successfully run the traffic capture. Notably, our architecture is able to handle traffic capture of multiple devices simultaneously. The captured traffic is then processed by the Analysis System (in our prototype, hosted on the capture server itself) to produce MIRAGE-2019.

In detail, the MIRAGE architecture builds the dataset in three phases as shown in Fig. 2.1b:

1. *Capture phase*: traffic traces are collected in PCAP format together with `strace` log-files keeping track of network system-calls.

2. *GT building phase*: PCAP traces are segmented and log-files are fil-

tered; this information is then combined to produce labeled traffic objects.

3. *Dataset extraction phase:* labeled traffic objects are processed to extract the information of interest (including statistical features) to be used as input data for exploratory analysis or ML tasks.

The next subsections provide details about these three phases.

2.2.1 Capture Phase

A capture session begins when an experimenter connects a mobile device to the USB hub: this procedure automatically kicks off the capture of the network traffic as well as the logging of the system calls. In detail, the traffic is captured on the wired interface of the capture server by means of `tcpdump` [119]. Leveraging traffic filters based on the MAC address of the connected devices allows to collect the traffic generated by multiple devices at the same time, without any ambiguity. On the other hand, the system-call tracing is enabled on the mobile device itself by using the `strace` utility via ADB [120]. In detail, `strace` is set to log network-related and process-management system calls (e.g., `connect`, `bind`, `getsockname`, `fork`, `wait`, `exec`, etc.), also logging the related `<IP:port>` pairs and associating each socket descriptor to the name of the Android package which originates the call.[2] As a result, this phase provides an `strace` log-file with GT information for each PCAP trace collected (see Fig. 2.1b). This kernel-based mechanism for gathering information being given, the finest TC-granularity we can achieve is the (bi)flow level.

[2]To this aim, we monitor the Android *zygote process* that handles the forking of each new application process. Then we extract the Android package names from the PIDs returned by the `fork` system calls.

The described capturing system allows the capture of mobile-app traffic when accessing the Internet through a Wi-Fi channel. In principle, the behavior of the apps could be different from the one shown when connecting through 3G/4G channel. Indeed, Android applications may detect the type of network that is used to transport their data, telling apart Wi-Fi, Mobile, and VPN connections and potentially acting in different ways according to that.[3] Other mobile-app traffic capture approaches leverage an encrypted connection (*i.e.* VPN) that tunnels the traffic over 3G/4G to a gateway/capture server [112]. Compared to our capture method, the VPN-based ones not only suffer from the identical issue (*i.e.* apps are able to know if the transport network is a VPN the same way), but possibly add uncontrolled changes to the statistical properties of captured traffic due to the underlying tunneling protocol mechanics and the network between the mobile device and the capture gateway, resulting in a traffic timing less accurate and possibly traces less representative of plain 3G/4G setups.

2.2.2 GT Building Phase

In the GT building phase, first the PCAP traces are segmented to obtain *traffic objects*. Then, each of these objects is labeled taking advantage of the information extracted from the associated `strace` log-file.

PCAP traces are segmented into *biflows*. As opposed to common heuristics leveraged to define the 5-tuple that identifies each biflow (cf. §1.3.1), we are able to order the items constituting the 5-tuple based on the knowledge of IP addresses of the Android devices used for the captures. Therefore,

[3] The `ConnectivityManager` method `getType()` returns, among the others, `TYPE_WIFI`, `TYPE_MOBILE`, `TYPE_VPN` according to the active connection type (see https://developer.android.com/reference/android/net/ConnectivityManager for API level before 21, or https://developer.android.com/reference/android/net/NetworkCapabilities for the analogous constants prefixed with `TRANSPORT_` in API level 21 and following).

in the dataset the addresses in biflows are ordered in the upstream direction (i.e. local-to-remote, with reference to the mobile terminal). Note that biflow segmentation in addition to being a common choice for traffic objects, it perfectly fits the metadata extracted from the network (viz. socket) system-calls.

Then, each biflow is labeled with the Android package-name that *exactly* matches the 5-tuple in the `strace` log-file, considering `getsockname` and `connect` system calls. In the case a perfect match cannot be found in the log-file for some biflows[4], the procedure assigns labels according to a heuristic, i.e. labeling these biflows with the *most-common label* (*i.e.* package name) in the PCAP trace. As this approach can potentially cause mislabeling, the case of heuristic labeling is explicitly marked as such in the dataset, allowing the final user to decide between considering in the GT all traffic objects (possibly noisy) or keep only strict matching ones.

2.2.3 Dataset Extraction Phase

This phase is in charge of taking each labeled traffic object and extract the relevant information to feed any potential application of the collected data (*e.g.*, exploratory mobile-app traffic analysis or ML algorithms to solve specific tasks). The output of this phase constitutes the final MIRAGE-2019 dataset. Since the traffic objects here considered correspond to biflows, information can be drawn in the form of summarizing statistics from the whole traffic object, or from a subset of the constituting packets. The specific types of information provided for MIRAGE-2019 and the related context are described in full details in the following section.

[4]This could be caused by either the pre-existence of these biflows before the capture started or by a failure of the `strace` in following the corresponding child-processes forked by zygote.

2.3 The MIRAGE-2019 Dataset

We have collected the MIRAGE-2019 dataset in the ARCLAB laboratory at the University of Napoli "Federico II". The capture sessions span from May 2017 to May 2019. We employed three devices to generate the mobile traffic, namely:

- Xiaomi Mi5
- Google Nexus 7
- Samsung Galaxy A5

In detail, we installed the custom firmware CyanogenMod v13.0 (corresponding to the Android version 6.0.1) on all the devices and enabled the root mode.

More than 280 experimenters took part to the dataset construction on a voluntary basis, by performing one or two experimental sessions each. The experimenters involved in this activity were students of three different courses[5] held at the University of Napoli "Federico II", aged $19 \div 25$ years, with a 85/15% share between males and females. Each experimental session lasted two hours, at most. Altogether, during each experimental session, every experimenter performed 12 capture sessions of $5 \div 10$ minutes, each resulting in one PCAP traffic trace and one `strace` log-file, as described in the previous section. In each capture session the experimenter was asked to perform activities mimicking common uses of a single app with the intent to explore its functionalities. In addition, we asked the experimenter to carry out:

[5]Namely: Computer Architectures, Computer Networks, and Internet Analysis and Performance.

- First-time installation of the app to be exercised (only for the first capture session).
- Registration of a new app-user (where possible).
- Login of the registered app-user (for half of the capture sessions, in contrast to already logged-in scenarios).

We report the ethical considerations underlying the aforementioned traffic-capture procedure in Appendix A.

MIRAGE-2019 is focused on apps running on Android, currently retaining the 76% of the whole market share [121]. Overall, the MIRAGE-2019 dataset gathers the traffic generated by 40 Android apps belonging to 16 different categories according to Google Play apps distribution portal [122]. Before each experimental session, the exercised app is updated to the latest version available on the Italian Play Store. The MIRAGE website (http://traffic.comics.unina.it/mirage) reports the detailed app metadata together with the links to their pages on Google Play. As a whole, a total of 4606 PCAP traces were collected within MIRAGE-2019.[6]

Figure 2.2 shows the cumulative distribution of the duration of the traces collected, having an average duration of 370 s. It can be noted that the majority of traces has a duration corresponding to that prescribed for capture sessions (*i.e.* $5 \div 10$ min.), being the median equal to 329 s and the 5- and 95-percentile equal to 213 s and 674 s, respectively. Differently, outliers are due to unintended disconnections from the traffic capture system, erroneous procedures carried out by the experimenters, but also specific experimental scenarios (*e.g.*, prolonged video-playing, calls, etc.).

[6] We have filtered out the PCAP traces having an `strace` log-file less than 200 kB or a duration less than 10 seconds.

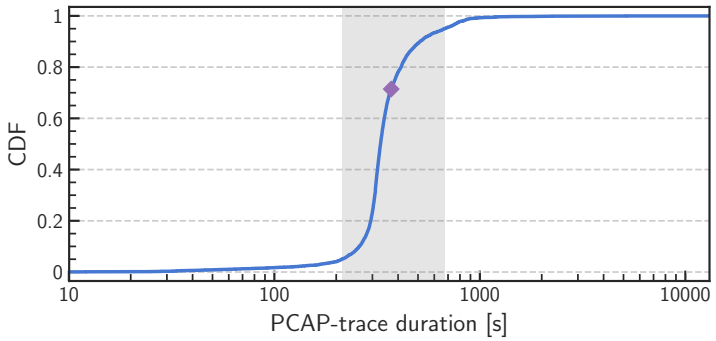


Figure 2.2: Cumulative distribution of the duration (s in log-scale) of the PCAP traces collected. The diamond marker reports the average, whereas the shaded box highlights the $(5, 95)^{th}$ percentile region.

We release the MIRAGE-2019 dataset in JSON format to foster its compatibility and increase its usability: one JSON file corresponds to one PCAP trace captured (*i.e.* a self-contained capture session). In detail, for each biflow—identified by its 5-tuple—we have extracted:

- *Per-packet data*
- *Per-flow features*
- *Per-flow metadata*

Figure 2.3 shows the structure of each JSON file. In the following, we provide the details about released data and their format, summarized also in Tab. 2.2.

Per-packet Data. We extract 6 informative header fields and the L4 payload of the first 32 packets of each biflow. Table 2.2a describes the data D_i extracted. In detail, each D_i identifies a list of up to 32 elements.

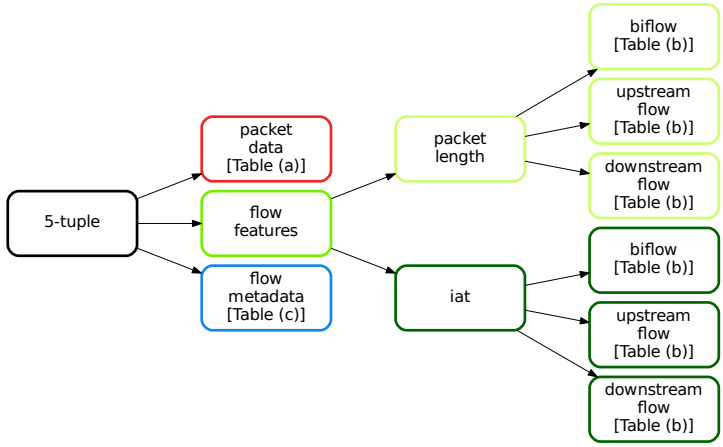


Figure 2.3: Structure of the JSON files constituting MIRAGE-2019.

Researchers performing TC [81, 94, 91] and WF [25] via ML and DL used these inputs in their works.

Per-flow Features. To provide information on the whole biflow and corresponding upstream and downstream flows, we select 17 statistical features computed on the sets of upstream, downstream, and complete (*i.e.* both of them) IP packet lengths and inter-arrival times, for a total of 102 per-flow features. Table 2.2b reports the per-flow features F_i extracted. Previous works in the field of TC via ML successfully leveraged these features to feed the classification algorithms they devised [76, 85, 43].

Per-flow Metadata. Table 2.2c describes per-flow metadata complementing per-flow features, being also related to complete biflow and upstream/-downstream flows. GT (viz. the biflow-label extracted) granularity (*i.e.* exact or most-common) refers to the GT-building procedures described in Sec. 2.2.2.

Table 2.2: The MIRAGE-2019 dataset. D_i, F_i, and M_i report the dict-keys in the released JSON files.

(a) Per-packet data extracted from the first 32 packets of each biflow.

D_i	Description
src_port	Source transport-layer port
dst_port	Destination transport-layer port
packet_dir	Packet direction (0 upstream, 1 downstream)
L4_payload_bytes	Number of bytes in L4 payload
iat	Inter-arrival time
TCP_win_size	TCP window size (0 for UDP packets)
L4_raw_payload	Byte-wise raw L4 payload (integer $\in [0, 255]$)

(b) Per-flow features extracted from the sets of upstream, downstream, and complete IP packet lengths and inter-arrival times.

F_i	Description
min	Minimum
max	Maximum
mean	Arithmetic mean
std	Standard deviation
var	Variance
mad	Mean absolute deviation
skew	Unbiased sample skewness
kurtosis	Unbiased Fisher kurtosis
q_percentile	q^{th} percentile ($q \in [10 : 10 : 90]$)

(c) Per-flow metadata M_i related to the complete biflow (`BF`) and upstream (`UF`) and downstream (`DF`) flows.

M_i	Description
BF_label	Android-package name
BF_labeling_type	Exact or most-common labeling
{*BF,UF,DF*}*_num_packets*	Number of packets
{*BF,UF,DF*}*_IP_packet_bytes*	Total bytes in IP packets
{*BF,UF,DF*}*_L4_payload_bytes*	Total bytes in L4 payloads
{*BF,UF,DF*}*_duration*	(Bi)flow duration in seconds

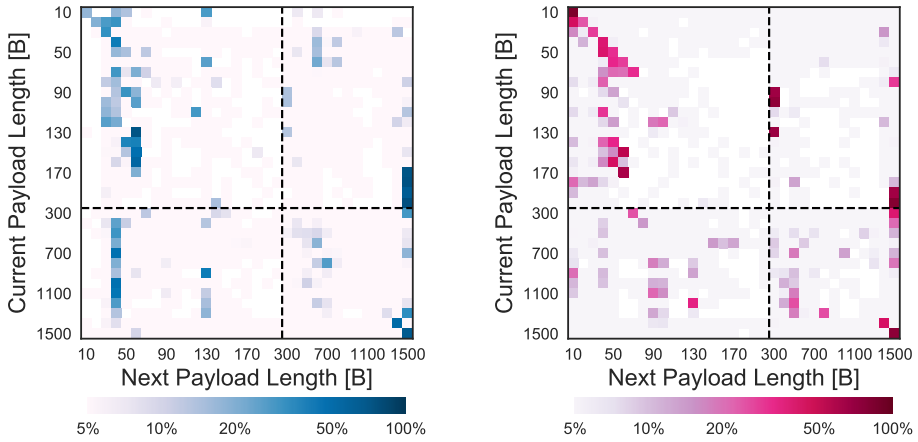
2.4 Example traffic-analysis tasks enabled by MIRAGE-2019

The released MIRAGE-2019 dataset can suit different kinds of traffic-analysis tasks, ranging from app traffic modeling and prediction, to biflow-based TC. Herein, we present an app-traffic characterization that can be directly derived from the dataset, concerning per-packet data (§2.4.1), per-flow features (§2.4.2), and per-flow metadata (§2.4.3). While providing a detailed characterization of MIRAGE-2019, our analyses also point at tasks enabled by the released dataset.

2.4.1 Per-app Modeling based on Per-packet Data

Modeling network traffic represents a key task in the study and design of Internet architectures, as realistic (yet manageable) traffic models are needed to predict, interpret, and solve performance-related issues of current and future networks.

To prove the suitability of MIRAGE-2019 to support this class of tasks, we provide a preliminary study aimed at devising *per-app* traffic models by means of Markov Models, similarly as done in [123]. In detail, for each app we consider the sequence of the `L4_payload_bytes` of each biflow (*i.e.* focusing on the first 32 packets, cf. §2.3) and derive the so-called *transition matrix*, whose $(X, Y)^{th}$ entry represents the probability that the next packet comes with Y bytes of payload if the last observed packet has a payload of X bytes (save from rounding errors due to binning). Notably: (*i*) packets with null payload are filtered out (*i.e.* only the packets transferring contents generated by the application layer are retained, while signaling such as TCP SYNs, RSTs, and *pure* ACKs are discarded, as not of interest for this analysis); (*ii*) the payload-size interval between 1 B and 1500 B is divided in 33



(a) com.dropbox.android. (b) air.com.hypah.io.slither.

Figure 2.4: Transition matrices of payload lengths. Axes are divided in two different linear scales. The color-bar is in log-scale.

non-uniform bins, namely: bins from 1 to 20 (resp. from 21 to 33) are 10 B (resp. 100 B) wide. This scheme allows to better appreciate the model dynamics by mitigating the impact of the high presence of packets with null or very-small payload.

As an example, Fig. 2.4 reports the transition matrices for *Dropbox* (Fig. 2.4a) and *Slither.io* (Fig. 2.4b). By looking at the two matrices, the following observations can be derived:

- The presence of high values (darker colors) along the diagonal witnesses the tendency to remain in the same state, *i.e.* sending/receiving consecutive packets of similar sizes. This observation holds for both apps, and becomes evident for very-small values (top-left corner) as well as for very-large values (bottom-right corner).

- Vertical patterns highlight the trend in entering to a specific state,

whichever the current payload length. This is particularly evident for Fig. 2.4a, where next expected payload length is within 30–40 B when current payload length lies within 300–1400 B, and in both apps when the current payload belongs to 1400–1500 B.

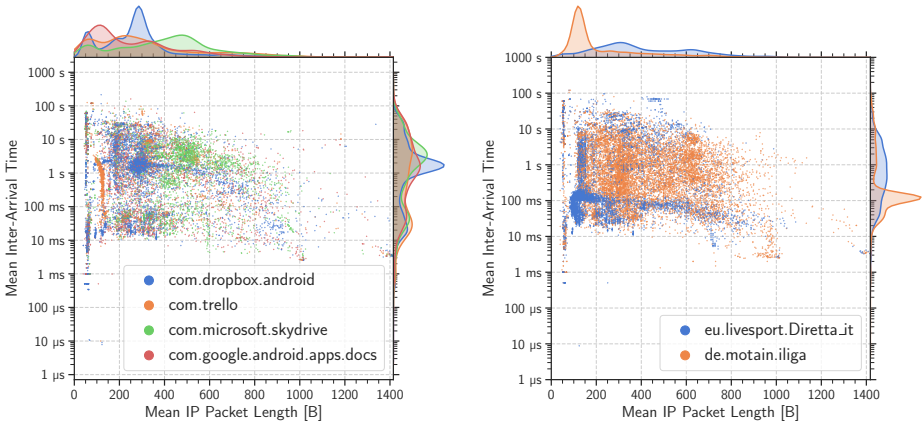
- Scattered darker points highlight app-specific patterns.

2.4.2 Per-flow Statistical Characterization

Characterizing mobile-app traffic based on its statistical features is a capability of the utmost importance that mitigates a number of issues and benefits different tasks in network administration. As discussed in Chapter 1, trends in network applications and protocol design (*e.g.*, protocol encapsulation, encrypted transmission, use of non-standard ports, concerns about users' privacy) heavily challenge TC when exploiting some of the developed techniques (*e.g.*, port-based and DPI). In fact, approaches based on statistical properties of network traffic provide viable alternatives (cf. §1.3).

Leveraging the information directly provided by MIRAGE-2019, the mobile-app traffic can be modeled at different granularities, considering both the apps and the related categories. As an interesting example, Fig. 2.5 reports the joint scatter plot (per biflow) of the mean packet lengths and inter-arrival times for four different app categories: `Productivity` (Fig. 2.5a), `Sports` (Fig. 2.5b), `Games` (Fig. 2.5c), and `Music & Audio` (Fig. 2.5d). Each figure reports the apps with different colors, while the kernel density estimation of the marginal distributions is shown in the side plots. Several considerations can be drawn from this analysis.

First, given the different spatial concentration of the points over the plane, different categories result in different joint scatter plots. For instance, while points in `Games` mostly lie within $[50, 400]$ B (x-axis) and $[10\,\text{ms}, 100\,\text{s}]$



(a) Productivity.

(b) Sports.

(c) Games.

(d) Music & Audio.

Figure 2.5: Joint scatter-plot of mean (payload length, inter-arrival time) of each biflow for four different app categories. The packet length is reported in linear scale (x-axis), whereas the inter-arrival time is shown using a log-scale ($10 \log_{10}(x)$).

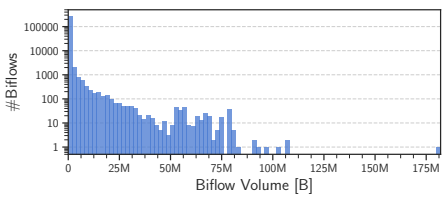
(y-axis), the same *does not apply* for the other three categories.

Secondly, the apps within the same category often show their own peculiar statistical profile. For example, *OneFootball* (`de.motain.iliga`) updates (Fig. 2.5b) generate points mostly concentrated around 100 B and 100 ms, as opposed to *Diretta* (`eu.livesport.Diretta_it`). Differently, for `Games` (Fig. 2.5c), while the inter-arrival time does not hold great discriminating power, the packet length allows to separate biflows associated to the two considered games easily enough.

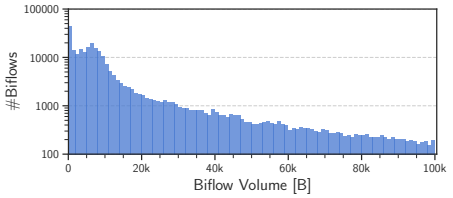
Finally, this rationale is not as much evident for the apps within `Productivity` and `Music & Audio` categories (see Fig. 2.5a and Fig. 2.5d). However, in the former case, *Dropbox* (`com.dropbox.android`) may be distinguishable based on peculiar profile of both mean packet lengths and inter-arrival times; whereas, in the latter case, *Spotify* (`com.spotify.music`) and *Musixmatch* (`com.musixmatch.android.lyrify`) show distinctive marginal distribution of only packet lengths and inter-arrival times, respectively. Indeed, the presence of traffic generated by several (similar) apps is one of the main challenges of mobile-app traffic analysis. This results in very-complex patterns of current traffic which cannot be appropriately captured by common statistical features. This outcome confirms the recent trend discussed in Sec. 1.2, applying novel ML- and DL-based techniques foreseen to be the effective workhorse to cope with mobile-app traffic challenges.

2.4.3 Per-flow Volume Distribution

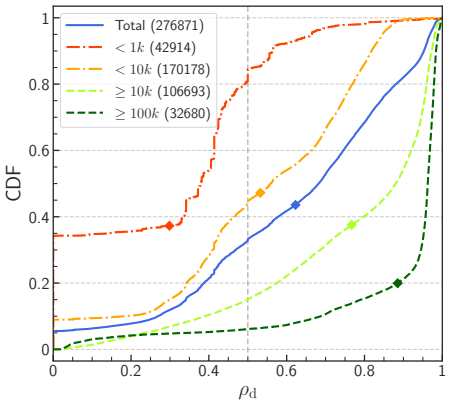
The information in MIRAGE-2019 also enables a characterization of the traffic based on the transferred volumes. Figure 2.6a shows the volume of the biflows in MIRAGE-2019, reporting the histogram of their sizes (bin width: 2 MB) which range from few bytes to several megabytes. In general terms, our collected information shows that request-response interactions are more



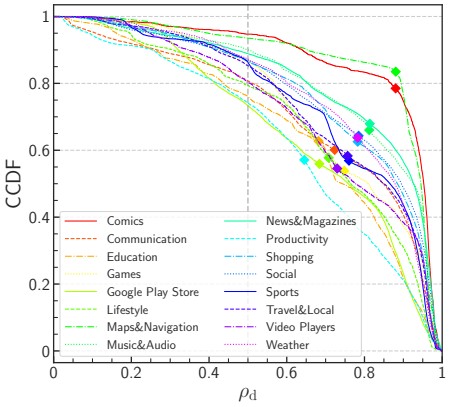
(a) Volume histogram of all biflows (bin width: 2 MB).



(b) Volume histogram of biflows with volume $\leq 100\,\text{kB}$ (bin width: 1 kB).



(c) Cumulative distribution of ρ_d. Diamonds mark the average.



(d) Complementary cumulative distribution of ρ_d for each app category (biflows with volume $\geq$ 10 kB). Diamonds mark the average.

Figure 2.6: Per-flow characterization of MIRAGE-2019 biflows in terms of byte volume (a-b) and downstream volume share ρ_d (c-d).

common than long data transfers. Accordingly, voluminous biflows are less frequent than (possibly short-lived) biflows transferring a limited amount of bytes. As more than 88% of the biflows have volumes ≤ 100 kB, Fig. 2.6b reports the histogram (bin width: 1 kB) for such dataset portion, revealing that the biflows with volumes within $[5, 10]$ kB are the most common.

To deepen the nature of the exchange, Fig. 2.6c reports—in the form of cumulative distributions—the *share* ρ_{d} of the downstream bytes for each biflow (*i.e.* $\rho_{\mathrm{d}} \triangleq \frac{B_{\mathrm{d}}}{B_{\mathrm{d}}+B_{\mathrm{u}}}$, where B_{d} and B_{u} are the number of downstream and upstream bytes of the biflow, respectively) for the whole MIRAGE-2019 dataset ($\approx$ 270k biflows). On average, downstream traffic accounts for $\approx 65\%$ of the volume of the overall biflow traffic. However, we can notice a clear pattern when considering biflows with different volumes: the smaller the biflow volume, the lesser the share of downstream traffic (*i.e.* $\rho_{\mathrm{d}} \leq 0.5$). Indeed, for biflows very small in volume (*i.e.* ≤ 1 kB) $\rho_{\mathrm{d}} = 0.35$ on average. Notably, for around 5% of the biflows no downstream packets were captured. As expected, all these communications are very small in volume (*i.e.* ≤ 10 kB) and are the results of anomalous conditions, with the mobile app asking for services to external servers and no reply returned due to failures possibly at the communication endpoint or along the path. Differently, when only flows with larger sizes are retained (*i.e.* ≥ 10 kB and ≥ 100 kB), the downstream traffic accounts for most of the volume, namely mean ρ_{d} equals to 0.77 and 0.89, respectively.

The analysis of the exchanged traffic volumes and (un)balance of the downstream-upstream ratio has an impact on access link dimensioning and per-app traffic volume costs (for the operator), and transmission power consumed (for the user). Indeed, breaking the above results down by different app categories (see Fig. 2.6d), we are able to quantify the expected difference of transmission resources required. According to this analysis, two

categories (`Comics` and `Maps&Navigation`) show a distinct distribution of ρ_{d} from the other ones, having a higher mean value and only $\approx 5\%$ of the biflows with $\rho_{\mathrm{d}} \leq 0.5$.

2.5 Benchmarking Traffic Classification

This section carefully describes the datasets used to assess the set of TC methodologies proposed in the present dissertation. These correspond to three human-generated mobile traffic datasets—encompassing part of MIRAGE-2019—and to the public Anon17 dataset [44], whose details are reported in Secs. 2.5.1 and 2.5.2, respectively.

2.5.1 Mobile Traffic Datasets

Leveraging the MIRAGE architecture, we collected mobile-app traffic for more than two years (starting from 2017), cross-sectionally and simultaneously to the research activities described in this Thesis. Meanwhile we have continued the experimental activities and updated the MIRAGE-2019 dataset, for benchmarking the set of devised TC methodologies, we have employed a self-contained part of MIRAGE-2019, that we have complemented with other two mobile *human-generated* datasets. In detail, the MIRAGE-2019 subsample encompasses the traffic pertaining to either *Facebook (FB)* or *Facebook Messenger (FBM)* apps, whose collection has been recommended by a global mobile solution provider; whereas the other two mobile datasets—named *Android* and *iOS*—have been produced and handed to us directly by the same provider[7].

We would emphasize that all the works dealing with mobile TC mostly

[7]Due to NDA with the provider, we can not report its name, details of its network, detailed information on the datasets, nor release them.

either consider iOS- [80] or (most of the time) bot-generated Android-traffic [43, 78, 84, 86] (see also Tab. 1.2). Conversely, the three datasets used herein cover both mobile operating systems with human-generated traffic. Indeed, the different nature and history of the software and hardware ecosystems revolving around the two operating systems suggest that app traffic could inherit different properties as well. Consequently, app discrimination ability on one system cannot be assumed as generalizable to the other.[8]

Table 2.3 summarizes the details of the three mobile traffic datasets we illustrate point-by-point in the following.

FB/FBM Binary Dataset. The *first (binary) dataset* was collected using the MIRAGE platform (viz. it is a subsample of MIRAGE-2019). In detail, we consider the capture sessions related to FB or FBM generated by means of the Xiaomi Mi5 (cf. §2.3) during May '17 - Mar. '18. This choice derives from the peculiar nature of these two apps, both devoted to interactive usage of the Facebook platform (author of both). This suggests a high possibility of shared development framework, overlapping services usage, and similar apps' fingerprints, hampering the discrimination of the respective traffic needed for key management tasks *e.g.*, *billing differentiation*. The capture sessions are performed as described in Sec. 2.2.1 and involved different activities (*e.g.*, posting contents, commenting, liking, sending messages, making (video-)calls, etc.) carried out by the experimenters to explore app diversity (cf. §2.3). We collected more than 1100 traffic traces and labeled them following the procedure reported in Sec. 2.2.2. The whole dataset contains $\approx$ 34.2k instances, with 15.0k (resp. 19.2k) biflows from

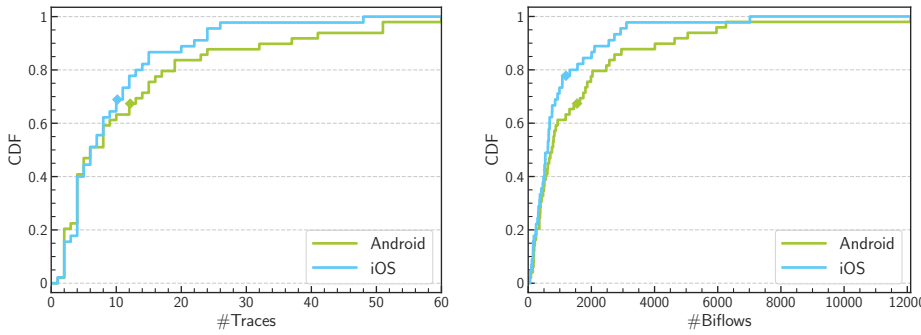
[8]Similar concerns regard bot-generated traffic when extending results to actual human-generated mobile app traffic.

Table 2.3: Details of the mobile traffic datasets employed in the experimental evaluation of TC methodologies presented in this Thesis.

Dataset	Type (#Apps)	#Traces	#Biflows	%ET	OS Version	Collection	Source
FB/FBM	Binary (2)	> 1100	34.2k	91%	Android 6.0.1	05/17 - 03/18	MIRAGE
Android	Multi-class (49)	607	77.3k	47%	4.2.2 - 6.0.1	04/15 - 01/17	Provider
iOS	Multi-class (45)	419	44.1k	60%	7.0 - 10.0	09/14 - 01/17	Provider

FBM (resp. FB) app and a 44%/56% share. Precisely, FBM (resp. FB) traffic consists of 13.2k (resp. 18.7k) TCP and 1.8k (resp. 0.5k) UDP biflows, respectively. The encrypted biflow ratio (%ET) corresponds to 91%.

Multi-class Android and iOS Datasets. The *other two (multi-class) datasets*, obtained from a global mobile solutions provider and generated from 49 (resp. 45) apps on Android (resp. iOS) devices, are explored for *prioritization purposes*. The corresponding Android (resp. iOS) traces have been collected during Apr. '15 - Jan. '17 (resp. Sept. '14 - Jan. '17), generated by users with different devices and OS/app versions, and provided already anonymized and cleaned from background traffic. Uniformly to the capture phase of MIRAGE, provided traces capture the traffic generated by users running a single app at a time on a given device/OS, thus limiting the presence of background traffic and allowing to label the traces with the associated known GT. In detail, $\approx$ 89% (resp. $\approx$ 85%) of Android (resp. iOS) traces has been captured in 2016. As a whole, the dataset is made up of 607 (resp. 419) traffic traces, with an average duration of 282 (resp. 296) seconds and 1 to 60 (resp. 1 to 48) traces per app in Android (resp. iOS). Moreover, 77.3k (resp. 44.1k) labeled instances compose the Android (resp. iOS) dataset, with 73.8k (resp. 41.8k) TCP and 3.5k (resp. 2.3k) UDP biflows and 47% (resp. 60%) encrypted biflow ratio (%ET).



(a) Cumulative distribution of #Traces. (b) Cumulative distribution of #Biflows.

Figure 2.7: Cumulative distribution of the number of PCAP traces and biflows per app for Android and iOS datasets. Diamonds mark the average.

Table 2.4 lists the apps belonging to both Android and iOS datasets, the categorical class-labels used to identify them in the following chapters, the number of biflows, and the number of PCAP traces per app. We can notice that the multi-class datasets exhibit a non-negligible class (viz. app) imbalance, both in terms of number of PCAP traces and biflows. This is even more evident by looking at their cumulative distributions reported in Fig. 2.7. The latter shows that the Android dataset has, on average, more per-app traces and biflows, being 12 and 1546, respectively, compared to 10 and 1194 of the iOS dataset. However, the long tail of the distributions discloses the presence of Android *PureVPN* (resp. iOS *SayHi*) app with up to 12k (resp. 7k) biflows, against the 75% of apps having less than $\approx$ 1850 (resp. $\approx$ 1100) biflows for the Android (resp. iOS) dataset. Such realistic and challenging setup allows a fair evaluation of proposed TC methodologies.

Further Observations. We underline that the number of instances also vary depending on the specific traffic object considered (other than bi-

Table 2.4: Apps pertaining to Android and iOS datasets. App name, categorical class-label (CCL), number of PCAP traces, and number of biflows are reported for each app.

Android				**iOS**			
App Name	**CCL**	**#Traces**	**#Biflows**	**App Name**	**CCL**	**#Traces**	**#Biflows**
360Security	1	2	178	360Security	1	2	158
6Rooms	2	41	2034	6Rooms	2	24	754
80sMovie	3	8	1639	80sMovie	3	6	1079
9YinZhenJing	4	3	178	Anghami	4	13	1081
Anghami	5	19	1869	AppleiCloud	5	20	3110
BaiDu	6	8	2567	BaiDu	6	8	2551
Crackle	7	4	426	Brightcove	7	8	989
EFood	8	2	352	Crackle	8	4	297
FrostWire	9	2	1187	EFood	9	3	102
FSecureVPN	10	5	257	FSecureVPN	10	5	153
Go90	11	5	752	Go90	11	6	372
Google+	12	37	1985	Google+	12	26	680
GoogleAllo	13	24	931	GoogleAllo	13	11	627
GoogleCast	14	2	145	GoogleCast	14	2	116
GoogleMaps	15	12	874	GoogleMaps	15	12	624
GooglePhotos	16	4	167	GooglePhotos	16	4	250
GooglePlay	17	60	5050	GroupMe	17	4	328
GroupMe	18	4	621	Guvera	18	4	517
Guvera	19	4	471	Hangouts	19	22	1739
Hangouts	20	16	500	HiTalk	20	5	541
HidemanVPN	21	19	2950	HidemanVPN	21	10	415
Hidemyass	22	2	373	Hidemyass	22	6	519
Hooq	23	17	1746	Hooq	23	4	462
HotSpot	24	23	4011	HotSpot	24	24	2729
IFengNews	25	13	1307	IFengNews	25	11	852
InterVoip	26	15	209	LRR	26	2	179
LRR	27	2	381	MeinO2	27	2	677
MeinO2	28	2	700	Minecraft	28	4	155
Minecraft	29	4	190	Mobily	29	2	58
Mobily	30	2	88	Narutom	30	4	930
Narutom	31	4	804	NetTalk	31	10	485
NetTalk	32	8	607	NileFM	32	7	652
NileFM	33	5	662	Palringo	33	8	650
Palringo	34	10	786	PaltalkScene	34	1	70
PaltalkScene	35	32	2473	PrivateTunnelVPN	35	15	2925
PrivateTunnelVPN	36	9	1444	PureVPN	36	14	1560
PureVPN	37	12	12107	QQReader	37	12	2007
QQ	38	51	6259	QianXunYingShi	38	4	539
QQReader	39	15	4633	Repubblica	39	7	1316
QianXunYingShi	40	4	551	Ryanair	40	2	319
RaidCall	41	8	390	SayHi	41	48	7019
Repubblica	42	6	830	Shadowsocks	42	4	238
RiyadBank	43	1	27	Sogou	43	4	760
Ryanair	44	2	512	eBay	44	15	2099
SayHi	45	51	5960	iMessage	45	9	359
Shadowsocks	46	4	1811	-	-	-	-
SmartVoip	47	6	246	-	-	-	-
Sogou	48	4	352	-	-	-	-
eBay	49	14	2731	-	-	-	-

flow) and relative segmentation parameters (*e.g.*, Service Burst—and Burst Threshold—employed for ML-based MC in Chapter 3). Moreover, it is worth noting that preprocessing operations might have been carried out on the datasets (both multi-class and binary), further varying the actual number of traffic objects (*e.g.*, required for the MIMETIC approach described in Chapter 5). If this is the case, we provide detailed report of traffic object statistics in the related chapters.

2.5.2 Anon17 Dataset

Anon17 was collected in a *real-network environment* at the Network Information Management and Security Lab [44] between 2014 and 2017 and gathers traffic from three anonymity tools: Tor [70], I2P [124], and JonDonym [125]. The dataset has been labeled leveraging the information provided by the anonymity tools themselves (*e.g.*, IP addresses of the Tor nodes) without relying on any application classification tool. The data are stored in ARFF format used in the data mining software tool *Weka* [126] and report the features (detailed discussed in later Sec. 4.3.2) either on a per-flow basis or pertaining to the Inter-Arrival Time / Payload-Length sequence of the first K packets of each flow. We point to [44] for obtaining exhaustive information on Anon17 dataset.

The main peculiarity of Anon17 is that it provides labels at three increasing-granularity levels:

(L_1) *Anonymous Network Level* (3 classes).

(L_2) *Traffic Type Level* (7 classes).

(L_3) *Application Level* (21 classes).

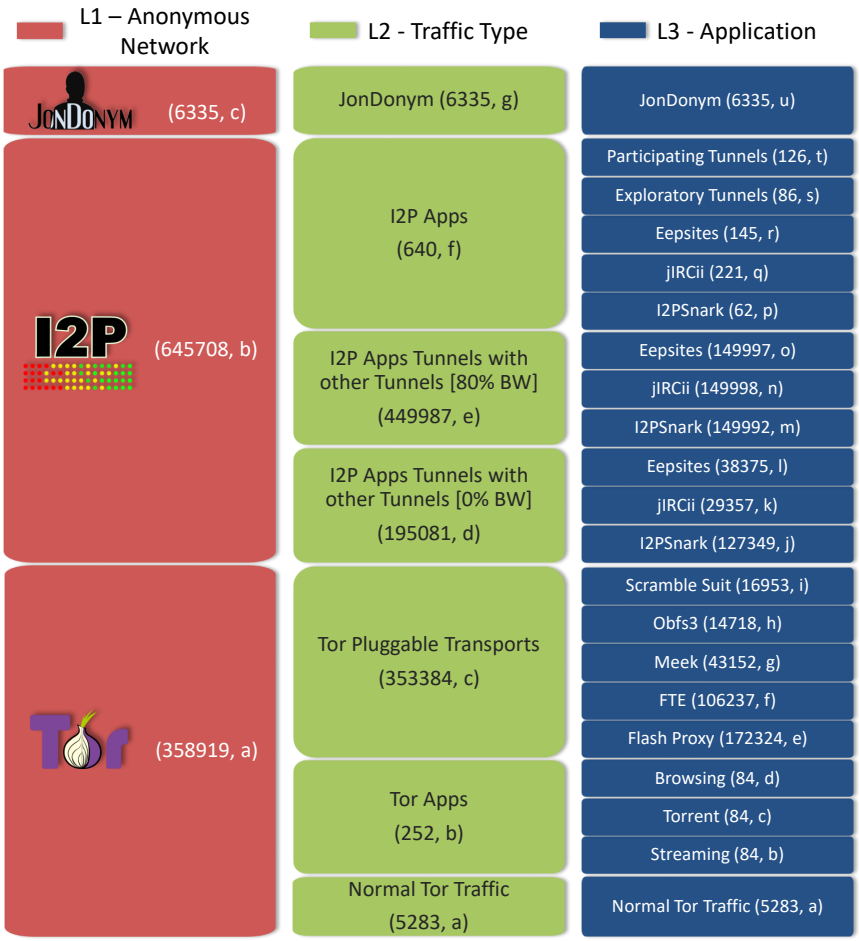


Figure 2.8: Anon17 Classification Levels: Anonymous network (L_1), Traffic Type (L_2) and Application (L_3), with total number of samples per class and class label at each level.

This label organization promotes the analyses of anonymous traffic at different levels, as well as the implementation of hierarchical approaches, as remarked in Fig. 2.8, reporting the categorization of Anon17 classes. Specifically, *Normal Tor Traffic* includes the circuit establishment and the user activities, whereas *Tor Apps* refer to flows running three applications on the Tor network (*i.e.* L_3 classes: *Browsing*, *Video streaming*, and *Torrent* file sharing). On the other hand, *Tor Pluggable Transports (PTs)* contain flows for five different obfuscation techniques (*i.e.* L_3 classes: *Flash proxy*, *FTE*, *Meek*, *Obfs3*, and *Scramble Suit*). The flows belonging to the L_2 class *I2P Apps Tunnels with other Tunnels* are collected by running three applications (viz. L_3 classes) on the I2P network: *I2Psnark* (file sharing), *jIRCii* (Internet Relay Chat), and *Eepsites* (websites browsing). The difference between 0% and 80% bandwidth is in the amount of sharing rate of the user bandwidth. *I2P Apps* contain traffic flows for the same three applications. However, in the latter case, management tunnels belong to separate L_3 classes (*i.e. Exploratory Tunnels* and *Participating Tunnels*). Lastly, *JonDonym* sub-dataset contains flows for the whole free mixes on the JonDonym network.

Anon17 exhibits a (majority) class imbalance problem, as shown by the total number of samples in Fig. 2.8 and summarily depicted in the top bar of Fig. 2.9. To cope with it, we have randomly down-sampled[9] (without replacement) by applying a pre-processing filter[10] to the instances of the following highly-populated traffic types, so as to keep their number comparable with the others: (*i*) Tor Pluggable Transports, (*ii*) I2P Apps Tunnels with other Tunnels [0% BW], and (*iii*) I2P Apps Tunnels with other Tunnels

[9]Over-sampling methods (*e.g.*, SMOTE, ROSE, etc.) are not considered here as Anon17 dataset does not show a minority class imbalance problem.

[10]Adopted filter is implemented in the Weka environment by means of `weka.filters.supervised.instance.Resample` Java class.

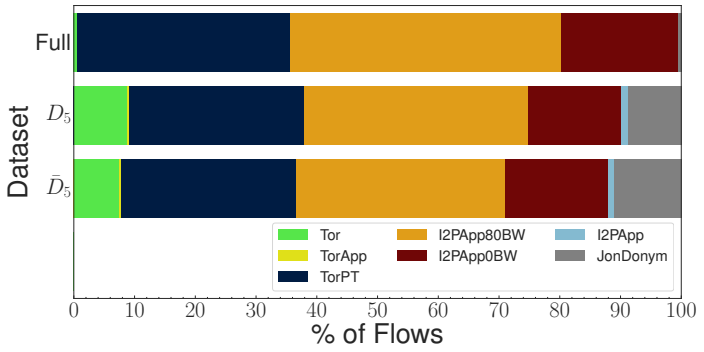


Figure 2.9: Down-sampling of Anon17 dataset: upper barplot (original *Full* dataset), middle barplot (down-sampling to 5%, D_5), lower barplot (removal of "all-zero-payload" flows, $\bar{D}_5$).

[80% BW]. The considered filter also preserves the proportions of the contained L_3 applications. Specifically, we consider a configuration corresponding to the down-sampling to 5% of the original dataset of each traffic-type set. For the sake of completeness, we have preliminarily tested two down-sampling configurations (corresponding to 5% and 10%) of each traffic-type set, showing a non-relevant difference in performance between them. However, aiming at a fairer investigation, here we have opted for the more balanced configuration. Figure 2.9 shows the percentage of flows labeled with different traffic types after performing the aforementioned down-sampling (middle bar D_5). We underline that we have chosen to down-sample the whole dataset, as opposed to the sole training set, since the latter choice would have biased the overall accuracy measure (evaluated from the test set) toward the performance of the majority classes.

Finally, to perform TC of sole informative flows, we have discarded from D_5 all the instances containing only zero-payload packets. The latter filtering procedure has been conducted implicitly by the inspection of the field

`maxPktSz` (reporting the maximum packet length of each flow), as Anon17 does not provide the complete sequence of packet lengths [44]. The bottom barplot in Fig. 2.9 shows the final resulting dataset (here denoted with $\bar{D}_5$) considered in the following flow-based TC. On the other hand, for the early TC, a specific different filtering procedure has been pursued, as explained in detail later in Chapter 4.

Chapter 3

Multi-Classification Approaches

As discussed extensively in Chapter 1, the traffic of mobile apps constitutes a moving target for classifiers due to its dynamic evolution and mix. Consequently, in this challenging scenario also standard ML-based classifiers may give way to this swift progression. Indeed, on the one hand, they are the most appropriate to deal with encrypted traffic, representing a large amount of mobile one [55], but on the other hand, naïve "standard" ML approaches might experience non-negligible performance degradation in the mobile context [127].

In this Chapter, we envision a "structural" improvement of ML-based traffic classifiers that aims to improve the classification performance of mobile apps by proposing a Multi-Classification System (MCS) which intelligently-combines decisions from state-of-the-art (base) classifiers specifically devised for mobile- and encrypted-TC and currently considered the best approaches in such context [43, 64, 65]. To the best of our knowledge, we perform this investigation in the mobile context for the first time (cf. §1.4).

Additionally, despite (wise) combination of state-of-the-ar t classifiers is here analyzed to show how current classification performance of mobile traffic can be improved, the proposed MCS is nor restricted to the considered set of classification algorithms and statistical features, neither to the operational scenario (*i.e.* classifiers for "early" TC—see §1.3.1—may be considered in the proposed framework without any further complication [128, 129]). Indeed, the MCS framework can potentially overcome the deficiencies of each single classifier not improvable over a certain bound, despite efforts in careful "tuning", and provide improved performance with respect to any of the base classifiers, also allowing for *modularity* of classifiers' selection in the pool, as each component may be readily plugged-in/out to improve performance further. For this reason, research has focused on MCSs in the last years [40, 72, 73, 74, 76].

Besides, with respect to the aforementioned works, our MCS allows for choosing from several types of combiners developed in the literature [130, 131] and based on both *hard* and *soft approaches*. These latter, in particular, have been successfully applied to many practical problems [130] and their application to mobile TC is deemed extremely appealing. It is worth noting that the different combiners employed in this Thesis constitute a wide spectrum of achievable performance, operational complexity, and training set requirements. Furthermore, the generality and the weak-coupling to any base classifier of the proposed MCS is capitalized to draw out "best practices" in mobile traces' pre-processing and (proper) traffic object segmentation.

Based on the mobile multi-class datasets of true users' activity described in Sec. 2.5.1, our results show that MC framework can improve classification performance with respect to the best base classifiers considered for the task. Specifically, it is shown that macro recall can be appealingly improved by

more than +9% on the best base classifier, and that there is room for further possible improvement with evidence of over +10% achievable by the ideal combiner. Finally, an investigation of subset selection of classifiers' pool, referring to all the combiners within the proposed MCS, is also reported, highlighting an additional path of improvement and possible complexity reduction.

In view of the streamlined contributions, this chapter is organized as follows. Sec. 3.1 details related works, whereas Sec. 3.2 collectively describes the considered MCS for encrypted mobile TC. More specifically, Sec. 3.2.1 recaps the traffic objects and introduces the set of features employed, whereas Sec. 3.2.2 describes the classification algorithms considered as base classifiers and Sec. 3.2.3 introduces the hard and soft fusion techniques adopted for their combination. Such detailed description is aimed at the full specification of the present approach, so as to enable easy implementation or porting to any architecture, and comparison with other approaches and tools. Experimental evaluation is reported in Sec. 3.3. We show both dataset pre-processing (§3.3.1) and discuss its implication on performance (§3.3.2); finally, we report the experimental results obtained with our MCS (§3.3.3).

3.1 Related Works

As shown in Tab. 1.2, TC of mobile apps has been object of huge interest by several recent works, mainly based on encrypted-traffic assumption. However, to the best of our knowledge *no previous work* tackled mobile TC using a MC framework. In Tab. 3.1, we sum up the main aspects of these works and provide a comparison with the MCS here proposed (last row). The first group reports the papers using MC approaches for TC, whereas the sec-

Table 3.1: Summary of previous works (by year) employing MC approaches (first group) and tackling mobile TC (second group). Only *our MCS* (last row) deals with mobile app traffic.

Paper	MT	ET	ML	MC	TO	Classes	Input Data	TC Technique	TC Performance
Szabo *et al.* [72]	○	○	○	●	BF	8 applications	Ports, payload, stats, connection patterns	WMV of S_1	≈ 99% best acc.
He *et al.* [73]	○	●	●	●	BF	8 applications	BC, PC, duration, rate, statistics of PS & IAT	Co-Forest	+11% acc. w.r.t. baselines
Callado *et al.* [74]	○	●	●	●	F/BF	≤ 12 applications	BC, PC, ports, duration, rate	RS, MV, DS, EDS of S_2	≥ 60% acc.
Dainotti *et al.* [76, 40]	○	●	●	●	BF	12 applications	Protocol, ports, duration, BC, statistics of PS & IAT, first 10 PS & IAT	MV, WMV, NB, DS, BKS, WER of S_3	≥ 97% acc.
Yoon *et al.* [82, 83]	○	○	○	●	BF	N/A	Header, payload, or statistical features	N/A	GT not available
Dai *et al.* [78]	●	○	○	○	H	6 Android apps	HTTP payload	DPI	GT not available
Stoöber *et al.* [41]	●	●	●	○	B	20 users	Statistics of PS & IAT	SVC, K-NN	≥ 90% acc.
Wang *et al.* [80]	●	●	●	○	B	13 iOS apps	Statistics of PS & IAT	RF	≈ 94% acc.
Taylor *et al.* [42, 43]	●	●	●	○	SB	110 Android apps	Statistics of PS	SVC, RF	86.9% acc.
Alan and Kaur [84]	●	●	●	○	T	1595 Android apps	First 64 TCP PS	WF	88% best acc.
Conti *et al.* [86]	●	●	●	○	T	≤ 11 actions	Clustering-based features	RF	95% best acc. / prec.
Saltaformaggio *et al.* [60]	●	●	●	○	SB	35 actions	Statistics of PC, PS, & IAT	SVC	78% prec. & 76% rec.
Li *et al.* [89]	●	○	●	○	H	12 Android apps	HTTP request & headers	VAE	99.6% acc.
Le *et al.* [97]	●	●	●	○	BF	7 apps	BC, PC, ports, & first 5 PS	NB, GBT, RF SVM, NN	99.5% acc.
Our MCS	●	●	●	●	*SB*	49 *Android* 45 *iOS apps*	*Statistics of PS*	*Soft/Hard combiners in §3.2.3*	+9.5% *rec. w.r.t. best classifier*

Mobile Traffic (MT). **Encrypted Traffic (ET)**. **Machine Learning (ML)**. **Multi Classification (MC)**.
Traffic Object (TO): biflow (BF), burst (B), flow (F), HTTP session (H), Service Burst (SB), TCP connection (T).
Input Data: byte count (BC), inter-arrival times (IAT), packet count (PC), packet sizes (PS).
TC Technique - *Classifier*: Bayesian Network (BN), Deep Packet Inspection (DPI), Gradient-Boosted Tree (GBT), K-Nearest Neighbors (K-NN), Multi Layer Perceptron (MLP), Naïve Bayes (NB), Neural Network (NN), Random Tree (RT), Random Forest (RF), Support Vector Classifier (SVC), Variational Autoencoder (VAE), WF (Website Fingerprinting).
Combiner: Behavior-Knowledge Space method (BKS), (Enhanced) Dempster-Shafer (EDS), Majority Voting (MV), Naïve Bayes (NB), Random Selection (RS), WERnecke's method (WER), Weighted Majority Voting (WMV).
S_1: Port-based, DPI, BLINC [45]. S_2: NBTree, PART, C4.5, BN, SVC. S_3: C4.5, K-NN, RT, Ripper, MLP, NB, Portload [53], port-based.
*N/A: information not available in the related manuscript.

ond those facing TC of mobile apps. In detail, we underline if they tackle (*i*) mobile and (*ii*) encrypted TC, possibly using (*iii*) ML-based classifiers and (*iv*) MC frameworks, the TC granularity (in terms of (*v*) traffic object, (*vi*) number and type of classes considered), (*vii*) the input data used to feed the classifier/combiner, and (*viii*) the specific TC techniques (being a classifier or combiner). Finally, we summarize (*ix*) the (best) classification performance obtained.

Traffic Classification via Multi-classification Approaches. Szabo *et al.* [72] have been the first proposing a combination method of multiple traffic classifiers that relies on different types of approaches leveraging various sources of information, namely statistics, payload (being thus not suitable for encrypted traffic), port, heuristics, and connection patterns. The final decision on application classification is given through a majority voting mechanism weighted by classifiers' priority, with payload-based one having the highest. Authors test their model on several network traces with a total of 8 different applications, and show that the proposed solution improves both the completeness and the accuracy of TC up to $\approx 99\%$ for certain applications, when compared to existing methods.

In [73], the authors propose *Co-Forest*, a ML-based TC model, which leverages ensemble learning, combining the predictions of multiple classifiers by voting, and semi-supervised co-training, utilizing both (a small number of) labeled and (a large number of) unlabeled samples. Such general framework is compared with both traditional ensemble learning methods (*i.e.* Random Forest and Bagging) and standard ML classifiers (*i.e.* C4.5 and Random Tree), trained/tested with a set of flow-based features (see Tab. 3.1) extracted from one-week traces captured at the edge of south campus of Sun Yat-Sen University in China. Experimental results (reported for

7 TCP applications plus the unknown class) show that Co-Forest can get $\approx 11\%$ error-rate decline after co-training process, on average.

Callado *et al.* [74] perform an evaluation of different algorithms for TC in four scenarios (*i.e.* active measurements, laboratory, academic, and commercial network), showing that the performance of the single method strongly depends on the context in which it is employed. Then, starting from these observations, they present generic classifiers' fusion rules (summarized in Tab. 3.1) and validate them in the same scenarios. Specifically, the authors consider various factors that could severely affect TC performance (*e.g.*, considering uni- or bi-directional flows, including bad-performing classifiers in the combiners' pool, etc.) and provide guidelines for the proper usage of combination algorithms. Extending the set of combiners considered, Dainotti *et al.* [76] apply the six combination methods reported in Tab. 3.1 to a set of eight different classifiers fed with both per-flow statistical features and input data suited for early-TC (cf. §1.3.1). Using a dataset collected at the University of Napoli Federico II, the authors demonstrate that the proposed MCS (implemented as classification plugins in the TIE platform [40]) can improve the overall accuracy over that of the best-performing classifier. This result is particularly significant in the case of early classification, showing that the accuracy decrease of the base classifiers can be effectively compensated by their combination.

Finally, Yoon *et al.* [82, 83] devise a multilateral TC framework based on four classification criteria: service, application, protocol, and function. Despite not being a "pure" MC approach, experimental results (related to Yahoo traffic) show that it is able to simultaneously identify the Yahoo-service provided, specific application or protocol, and traffic purpose. Authors claim that they were able to classify 99% of traffic, but the lack of a valid GT hampers the evaluation of TC performance.

State-of-the-art Techniques for Traffic Classification of Mobile Apps. Dai *et al.* [78] have firstly introduced the concept of "network profile", playing the same role as DNA profiles for an Android app (*i.e.* a network fingerprint). They propose *NetworkProfiler*, a system composed of a module automatically executing an app in an emulator (*DroidDriver*), and another module that from the generated network traffic builds a profile in terms of (*i*) contacted hosts and (*ii*) a state machine of string sequences in URLs (*Fingerprint Extractor*). Being based on DPI (*e.g.*, HTTP payload) features, the extractor is *not suited* for encrypted traffic. The approach has been shown to be effective in identifying ad-traffic, whereas for non-ad apps the evaluation has been carried out only for 6 apps. Additionally, in [78] the full ground truth of the traffic traces being analyzed is not available, so making it hard to quantify the classification performance of *NetworkProfiler*. A similar spirit permeates the review of Tongaonkar [127], where challenges and techniques for mobile TC and app identification are discussed, mainly based on signature generation and fingerprint extraction from mobile traffic payloads and apps' metadata, as well as from third-party services (*e.g.*, advertisement and profiling traffic). Nevertheless, the problem of dissecting encrypted traffic is there bypassed by considering man-in-the-middle solutions, suitable only in controlled environments such as enterprises. Li *et al.* [89] propose a DL classifier, based on Variational Autoencoders and input data taken from the reconstructed HTTP session and thus also designed only for *clear traffic*. Authors employ a self-generated dataset comprising the traffic of 12 Android apps and extract HTTP request lines and header fields that convert to "input image" data. In this setup, experimental results show an accuracy up to 99.6% using a censoring threshold on the classifier output (cf. §1.3.2). Unfortunately, the authors do not provide any information on neither threshold value nor percentage of censored samples.

Stöeber *et al.* [41] develop a fingerprinting scheme for devices by learning their traffic patterns through background activities. They contend that 70% of smartphone traffic belongs to background activities, and this can be leveraged to create a fingerprint. Based on 3G transmissions, *bursts* of data are considered to evaluate statistical features. Then, by means of Support Vector Classifier (SVC) and K-Nearest Neighbors (K-NN), a model of the traffic to be fingerprinted is built, being capable of identifying similar bursts. Results show that using ≈ 15 minutes of traffic testing (based on 6 hours of training) leads to an accuracy $\geq 90\%$ (among 20 users with different combinations of apps installed). Wang *et al.* [80] propose a system for classifying app usage over encrypted 802.11 traffic (reporting results for 13 iOS apps from 8 distinct categories). Data frames are collected from target apps by running them dynamically for 5 minutes and training a Random Forest (RF) classifier with the proposed set of features. The need for an accurate ground-truth labeling is raised, highlighted by a counterintuitive behavior of some app performance with the training time. *AppScanner* is proposed in [42] as a framework for fingerprinting and identification of mobile apps. The fingerprints are collected by running apps automatically on an Android device and the network traces are pre-processed (to remove background traffic and extract features) to train an SVC and an Random Forest (RF). Statistical features are collected on sets of packets defined through timing criteria and destination IP address/port (cf. §1.3.1 and §3.2.1). The results, evaluated on 110 most popular apps from Google Play Store, report 99% average accuracy in identifying single apps, and up to 86.9% in classifying them, outperforming state-of-the-art alternatives devised for the (conceptually-)similar WF issue [64, 65]. More recently, AppScanner has been employed on a larger dataset to test the aging of apps' fingerprints (due to updates) and possible invariance with respect to used device and

app versions (due to different users' usage) [43]. It is demonstrated that, though updates, time, and different devices lead to a performance degradation (with updates being the more demanding issue), a good classification accuracy can be still achieved. To this end, a method for the removal of background / third-party services traffic is there conceived, however not verified by an accurate labeling of the actual non-specific app traffic. The terms of comparison in [42, 43] are also used by Alan and Kaur [84] to investigate whether Android apps can be identified from their launch-time traffic using only TCP/IP headers (*i.e.* the sizes of the first 64 packets). They find that apps can be identified with 88% accuracy when training and test sets are collected on the same device, based on the simple classification methods developed in [64, 65]. On the other hand, accuracy drops significantly (up to 26% for the best classifier) when the OS/vendor is different. The same work analyzes the impact of the amount of training data required for classification and its "aging" (due to updates). It is worth noticing that the state-of-the-art approaches employed in the following as base classifiers (cf. §3.2.2) outperform those analyzed in [84] in terms of accuracy and also of other performance metrics (see §3.3.3). Le *et al.* [97] propose a framework to integrate various state-of-the-art ML algorithms, BD analytics platforms, software-defined networking, and network functions virtualization for 5G self-organizing network applications. As part of this framework, they implement five ML-based traffic classifiers (see Tab. 3.1) using as features byte and packet counts, source and destination ports, and sizes of the first five packets of each biflow. Among considered classifiers, RF shows the best performance, reaching 99.5% accuracy in the classification of 7 apps[1].

[1]Despite the authors claim that their framework can operate with both Android and iOS apps, they do not provide any information about the dataset used for the experimental evaluation.

Other works aimed at identifying fine-grained user actions within mobile-app traffic. Conti *et al.* [86] recognizes specific actions that users perform while running a certain app, based on packet direction/size info. This is achieved through the classification of incoming/outgoing/complete time-series obtained from TCP connections via RF approach fed with clustering-based features. Specifically, given an action, the k^{th} feature indicate the number of connections that have been assigned to the cluster C_k after the execution of that action. This approach leads to $\geq 95\%$ accuracy for most of the considered actions within a set of 7 Android apps. *Netscope* [60] performs a similar task taking into account a set of 35 different activities (for both iOS and Android devices), based on statistics originated from IP headers. Assuming an eavesdropper on a Wi-Fi network, it is shown that even a small portion of encrypted traffic is enough for a given app to be recognized. K-means clustering is employed for elementary-behavior discovery and then an SVC is trained/tested on activity-behaviors binary mapping, showing performance that varies with the device being tested, but reach 78.04% precision and 76.04% recall, on average.

3.2 Multi-classification System Architecture

Figure 3.1 graphically depicts the proposed MCS as a whole. Following its workflow, in next sections we describe the traffic objects adopted along with the definition of the features extracted from observed traffic (§3.2.1), the state-of-the-art approaches used as base classifiers (§3.2.2), and finally the hard and soft combiners employed for classifier fusion (§3.2.3).

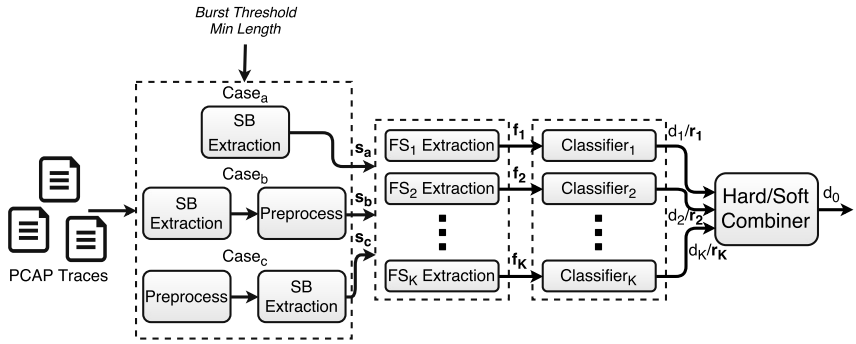


Figure 3.1: Architecture of the Multi-Classification System (MCS) proposed.

3.2.1 Traffic Object and Features

Traffic Object. In our MCS, the network traffic is decomposed into *service bursts (SBs)*, leveraging the notions introduced in [41] and [42, 43] for mobile-phone identification and mobile-app classification, respectively, and recapped in Sec. 1.3.1.[2]

Since the *burst threshold (BT)* is an essential parameter in the definition of SBs and a few different values have been chosen in recent studies [132, 42, 43], in our analysis we also account for sensitivity of classification performance to this parameter (see §3.3.2). The process of extracting the SBs from the considered traffic traces is summarized in Fig. 3.1 through the block *SB Extraction*. In Sec. 3.3.2, we will also investigate the need for a preprocessing step (represented in Fig. 3.1 as the *Preprocess* block) and, in affirmative case, whether this should be performed before or after the "burstification" process.

[2]We point that the SB notion has been used previously in [42, 43] under the (different) name of *flow* which is here used to refer to the common and established decomposition based on the direction-dependent 5-tuple (cf. §1.3.1).

Statistical Features. For the purpose of TC, we will consider features which are extracted by statistical means from the whole vector of packet lengths of the generic SB. This approach is analogous to flow-based TC when a flow (resp. a biflow) is instead considered as the relevant object of classification and the features are extracted from the sequence of packets forming it. It is worth mentioning that other feature sets may be considered, especially when early-TC of the generic SB is deemed of interest. The aforementioned class includes the packet sizes of the first K packets or some statistical features extracted from this "early" segment, as studied in [128, 129] for the case of standard Internet TC.

For each SB, three packet series are here considered: (*i*) *incoming* packets only (In), (*ii*) *outgoing* packets only (Out), and (*iii*) *bidirectional* traffic (*i.e.* both incoming and outgoing packets, In&Out). The following features can be identified for each of these series [42]:

- vector of packet lengths with sign indicating direction;
- minimum, maximum, mean, median, absolute deviation, standard deviation, variance, skew, and kurtosis of packet lengths;
- percentiles (from 10% to 90%, with 10% increments) of packet lengths.

Also, for the incoming and outgoing packet series taken as a whole, the joint histogram of packet lengths in both directions can be considered [64, 65].

Finally, in the following, the set of M features adopted by each classifier will be generically indicated with $f_1, \ldots, f_M$ (or collectively as $\boldsymbol{f} \triangleq \begin{bmatrix} f_1 & \cdots & f_M \end{bmatrix}^T$) and the set of classes (apps) as $\Omega \triangleq \{c_1, \ldots, c_L\}$.

The process of extracting the feature set for k^{th} classifier from each SB is summarized in Fig. 3.1 through the block FS_k *Extraction*.

Table 3.2: Summary of state-of-art techniques selected as base classifiers.

Abbreviation	**Method**	**Features Set**	**Reference**
`Lib_NB`	Naïve Bayes (NB)	Joint In&Out Histogram	[64]
`Her_Pure/TF/Cos`	Multinomial NB	Joint In&Out Histogram	[65]
`Tay_RF`	Random Forest	Stats + Percentiles (In/Out/In&Out)	[42, 43]
`Tay_SVC`	Support Vector Classifier	Stats + Percentiles (In/Out/In&Out)	[42]
`CART`	Decision Tree	Stats + Percentiles (In/Out/In&Out)	[133]

3.2.2 Base Classifiers

In this section we list the state-of-art approaches that we selected as the pool of $K = 9$ *base classifiers* employed in our MCS. The generic k^{th} base classifier within the considered pool is represented in Fig. 3.1 by means of the block $Classifier_k$. More specifically, this block receives as input the corresponding k^{th} *feature set* from the preceding feature extraction block and outputs either a *hard or soft decision* (mathematical details are later provided in Sec. 3.2.3) to the hard/soft combiner. We briefly describe their main properties and the motivations that guided us to their choice. For all of them we have reproduced their exact implementation and executed them with the same parameters as described in the respective works, to which we refer for further details.

A recap of the base classifiers considered in this paper, along with the abbreviations used, the supervised philosophy, the set of features taken as input, and the corresponding reference is given in Tab. 3.2.

Lib_NB. In Liberatore and Levine [64], two classifiers were proposed, one based on the Jaccard similarity index and another based on the Naïve Bayes (NB) learning technique. It was observed that the NB enjoys attractive performance and increased robustness than the Jaccard-based classifier, if IP packets are padded; thus we select the NB-based approach as a base clas-

sifier (`Lib_NB`). The NB assumes class-conditional independence of the features $\boldsymbol{f}$ that is not the case for real-world problems but working well in practice, and evaluates the probability that a test instance $\boldsymbol{f}_T$ belongs to each class c_i, *i.e.* the posterior probability $P(c_i|\boldsymbol{f}_T)$ through the Bayes' theorem $P(c_i|\boldsymbol{f}_T) \propto P(c_i) \prod_{m=1}^{M} P(f_{T,m}|c_i)$, where "$\propto$" denotes proportionality. The term $P(c_i)$ denotes the (prior) probability that a generic sample from the dataset will belong to c_i and is estimated from the training set population, while each PDF $P(f_{T,m}|c_i)$ is estimated by employing the (Gaussian) kernel density estimation. The fine-grained feature there employed is the joint histogram of packet lengths in both incoming and outgoing directions.

Her_Pure, Her_TF, and Her_Cos. Herrmann *et al.* [65] proposed the use of the Multinomial Naïve Bayes (MNB) classifier, adopting the same set of features as `Lib_NB` [64] but differing in the building assumption. Indeed, the NB classifier estimates each feature PDF using Gaussian kernels whose occurrence frequencies of the various packet sizes match best with the observed values in the test instance. On the other hand, the MNB classifier treats the f_ms as frequencies of a certain value of a categorical random variable and compares the sample histogram of each test instance with the aggregated histogram of all training instances per class. Then, the evaluation of the conditional PMF $P(\boldsymbol{f}_T|c_i)$ is different from `Lib_NB` and equals $P(\boldsymbol{f}_T|c_i) \propto \prod_{m=1}^{M} (\rho_m)^{f_{T,m}}$, where ρ_m denotes the probability of sampling the m^{th} feature. This implementation is referred to as `Her_Pure` in our analysis. A few variants of MNB classifier, adopting *term frequency transformation* without and with cosine normalization, were also successfully employed in [65] and compared in [42], and are referred in our analysis to as `Her_TF` and `Her_Cos`, respectively.

Tay_RF and Tay_SVC. In [42, 43], four (resp. two) approaches for mobile-app traffic classification (resp. identification) were proposed, leveraging both an SVC (only in [42]) and an RF. The SVC is a supervised model that represents the training samples as points in a feature (viz. vector) space, with the aim of finding a set of hyperplanes which provide the best class separation. Then, during the testing phase, the SVC classifies the new points according to the portion of space they fall into. On the other hand, the RF is an ensemble classification method taking advantage of several decision trees built at training time in order to form a stronger classifier obtained by combining the ideas of *bootstrap aggregating* and *random-feature selection* to avoid over-fitting [134].

In [42], these classifiers were fed with either (*i*) raw vectors of packet lengths or (*ii*) statistical features extracted from mobile-app traffic, with the latter approach leading to the best and least complex classifier (RF with statistical features employed also in [43]) between the two. The latter set has been drawn out in [42] as the most "informative" from a larger set of 54 statistical features (*i.e.* min, max, mean, standard deviation, variance, mean absolute deviation, skewness, kurtosis, and percentiles pertaining to upstream/downstream/complete IP packet length sequences) by means of feature selection technique on mobile traffic data. For this reason, we consider both RF (`Tay_RF`) and SVC (`Tay_SVC`) based on the 40 statistical features selected in [42].

CART. Several works performed TC by means of decision trees (*e.g.*, C4.5, C5.0, and their variants [135]), both as flat classifiers [136, 133] and also in a hierarchical [85] or multi-classification [74, 76] architecture. In this paper, we leverage the Classification and Regression Tree (`CART`), a very similar variant of the C4.5 algorithm, constructing binary trees exploit-

ing the features and thresholds that ensure the maximum information gain at each node and allowing to perform both classification and regression tasks (*i.e.* with categorical and numerical target variables, respectively). The above classifier is fed with the same statistical features as `Tay_RF` and `Tay_SVC`.

3.2.3 Classifier Fusion Rules

Different classifier fusion rules (viz. combiners) have been proposed in literature [76, 130]. In the following, we will focus firstly on *hard combiners*, relying on *Type 1 classifiers* (*i.e.* those that output only the predicted class). Then, we will discuss fusion rules resorting to classifiers' soft-outputs (viz. *Type 3 classifiers*), namely the *soft combiners*. The generic (hard/soft) combiner adopted within the proposed MCS is shown in Fig. 3.1 through the block *Hard/Soft Combiner*.

In the proposed MCS, we will consider both *non-trainable* and *trainable* combiners [130]. In the former case, the combiner has no extra parameters that need to be trained, that is the combiner is ready-to-use once the sole base classifiers are trained. In the latter case, the combiner requires some parameters to be estimated, usually by means of a *validation set*, different from both the training and the test sets. Overall, the proposed MCS will provide *twenty different choices*, namely 6 hard- and 14 soft-combiners, respectively, as the classifier-fusion block being employed.

Finally, for completeness of performance evaluation, in Sec. 3.3.3, we will also consider an ORAcle combiner (`ORA`), *i.e.* an ideal upper bound on the performance corresponding to a combiner correctly classifying a test sample if at least one of the base classifiers provides the correct decision.

Hard Combiners. Hard combiners are based on Type 1 classifiers, that is they exploit only the classifiers' predicted classes generically denoted with $\hat{d}_k(\boldsymbol{f})$ and collectively as $\hat{\boldsymbol{d}}(\boldsymbol{f}) \triangleq \left[\hat{d}_1(\boldsymbol{f}) \quad \cdots \quad \hat{d}_K(\boldsymbol{f})\right]^T$, implying the least requirements for designers [130].

In what follows, we will denote with $\mu_i(\hat{\boldsymbol{d}}_T)$ the confidence attributed to the i^{th} class by a generic hard combiner, based on decisions $\hat{\boldsymbol{d}}_T \triangleq \hat{\boldsymbol{d}}(\boldsymbol{f}_T)$ pertaining to the test instance $\boldsymbol{f}_T$. Then, the combiner decision is obtained as

$$\hat{d}_0 \triangleq \arg\max_{i \in \Omega} \mu_i(\hat{\boldsymbol{d}}_T).$$

Before proceeding, we recall the definition of k^{th} classifier confusion matrix $\boldsymbol{E}^k$, whose $(i,j)^{th}$ entry is denoted with $e^K_{i,j}$ and represents the probability of k^{th} classifier deciding for j^{th} class when the i^{th} class is being observed (cf. §1.3.2). Clearly, the matrices $\boldsymbol{E}_k$ employed by combiners are typically estimated using a validation set. This is true for the estimation of the prior class-probabilities $P(c_i)$ as well.

In this work, the following hard combiners[3] will be considered [130]:

1. *Majority Voting* (`MV`): the estimated class corresponds to the one voted by the relative majority of the classifiers. In case multiple classes obtain the same highest value, ties are broken either (a) randomly or (b) by using e^k_{ii}, *i.e.* the vote of each classifier is weighted by the confidence degree of that classifier when it assigns a sample to the class it is voting for [76]. In the latter case, the `MV` becomes a *trainable combiner*.

2. *Weighted Majority Voting* (`WMV`): this approach is an advancement of the `MV`, obtained by weighting the vote of each classifier by its relative

[3]Note that all the hard combiners considered here are trainable, except for the Majority Voting with random tie-breaking.

confidence. The i^{th} class confidence of the combiner is evaluated as:

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq \left\{ \delta_i + |I_+^i| \cdot \ln(L-1) + \sum_{k \in I_+^i} w_k \right\},$$

where I_+^i denotes the subset of classifiers having decided for i^{th} class, $\delta_i \triangleq [\ln P(c_i)]$ denotes a class-constant offset, and $w_k \triangleq \ln(p_k/(1-p_k))$ denotes the weight of k^{th} classifier, with p_k being the estimated accuracy [131].

3. *Recall Combiner* (`REC`): this combiner relaxes the assumption of equal class-conditional accuracy (viz. recall) in `WMV` and thus it amounts to different individual class-specific recalls. The `REC` confidence measure is then:

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq \left\{ \bar{\delta}_i + |I_+^i| \cdot \ln(L-1) + \sum_{k \in I_+^i} w_{k,i} \right\},$$

where I_+^i denotes the subset of classifiers having decided for i^{th} class, $\bar{\delta}_i \triangleq [\ln P(c_i) + \sum_{k=1}^{K} \ln(1-p_{k,i})]$ denotes a class-constant offset, and $w_{k,i} \triangleq \ln(p_{k,i}/(1-p_{k,i}))$ denotes the weight of k^{th} classifier when deciding for i^{th} class, with $p_{k,i}$ being its estimated class-conditional accuracy [131].

4. *Naïve Bayes* (`NB`): this combiner represents the i^{th} class confidence measure by the *a posteriori* probability $P(c_i|\hat{d}_1, \ldots, \hat{d}_K)$ based on the conditional independence of classifiers, that is:

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq P(c_i) \left\{ \prod_{k=1}^{K} P(\hat{d}_{k,T}|c_i) \right\}.$$

5. *Behavior-Knowledge Space method* (`BKS`): this approach removes the conditional independence assumption of `NB` combiner via multinomial counting on the joint classifiers' space $\hat{d}_1, \ldots, \hat{d}_K$ [137]. More specifically, the validation set is used to estimate the *a posteriori* probability $P(c_i|\hat{\boldsymbol{d}})$ for each c_i and for each value of $\hat{\boldsymbol{d}}$.[4] This allows labeling each possible value of $\hat{\boldsymbol{d}}_T$ with the most likely class, according to $\mu_i(\hat{\boldsymbol{d}}_T) \triangleq P(c_i|\hat{\boldsymbol{d}}_T)$ and constructing a look-up (BKS) table. Then, during the testing phase, each new $\hat{\boldsymbol{d}}_T$ provides an index to retrieve from the BKS table the estimated class $\hat{d}_0$. Ties are resolved by using a `MV` (with random tie-breaking) between the elements of $\hat{\boldsymbol{d}}_T$.

6. *WERnecke's method* (`WER`): `WER` constructs the same table as `BKS` but, to reduce over-fitting, considers the 95% confidence intervals of the frequencies in each unit calculated by adopting the normal approximation of the Binomial distribution. If there is overlap among the intervals, there is no dominating class for labeling the test instance $\hat{\boldsymbol{d}}_T$. In this case, the "least wrong" among the K classifiers is identified based on the confusion matrices and authorized to assign the class to that unit.

Soft Combiners. This section discusses the combiners based on *Type 3 classifiers* [130]. More specifically, we assume that k^{th} classifier is able to provide a soft-output vector $\boldsymbol{r}_k(\boldsymbol{f})$ collecting L degrees of support (each belonging[5] to the interval $[0, 1]$), whose i^{th} entry $d_{k,i}(\boldsymbol{f})$ denotes the confidence that k^{th} classifier gives to the hypothesis that $\boldsymbol{f}$ was generated from class c_i.

[4] The space complexity is thus $\mathcal{O}(L^K)$, which requires a large validation set for training.

[5] Such constraint corresponds to the natural range of the output of a confidence measure and can be ensured even though the specific classifier does not admit normalized soft-outputs, see [130].

Consequently, for a feature vector input $\boldsymbol{f}$ the outputs of a pool of K classifiers can be summarized in a $K \times L$ Decision Profile (DP) matrix, denoted with $\boldsymbol{D}(\boldsymbol{f})$. It is worth noting that k^{th} row of $\boldsymbol{D}(\boldsymbol{f})$ equals $\boldsymbol{r}_k(\boldsymbol{f})$, whereas i^{th} column of $\boldsymbol{D}(\boldsymbol{f})$, denoted with $\boldsymbol{d}_i(\boldsymbol{f})$, represents the soft-confidence attributed to i^{th} class by the classifiers' pool.

In what follows, we will denote with $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T))$ the confidence attributed to i^{th} class by the generic soft combiner based on the DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ obtained from the test instance $\boldsymbol{f}_T$. The corresponding decision is then found as:

$$\hat{d}_0 \triangleq \arg\max_{i \in \Omega} \mu_i(\boldsymbol{D}(\boldsymbol{f}_T)).$$

The soft-combiners can be mainly categorized into *Class-Conscious (CC)* and *Class-Indifferent (CI)* methods. CC methods use the DP matrix but disregard part of the information, using only *one column per class* (*i.e.* $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T)) = \mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$). For this class of soft combiners, there exist either trainable or non-trainable combiners. On the other hand, CI methods use the whole DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ to evaluate i^{th} class confidence, *i.e.* they interpret the DP as a vector in the intermediate feature space. Only trainable combiners belong to the CI category.

The following soft combiners have been considered in this work [130]:

1. *[CC] Non-trainable combiners*: the combination function can be chosen among different simple alternatives, such as:

   - `Mean`:
   $$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \frac{1}{K} \sum_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)$$
   - `Maximum`:
   $$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \max_k d_{k,i}(\boldsymbol{f}_T)$$

- `Minimum`:

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \min_k d_{k,i}(\boldsymbol{f}_T)$$

- `Median`:

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \mathrm{med}_k d_{k,i}(\boldsymbol{f}_T)$$

- `Trimmed Mean` (`Trim`) `Mean`: the K degrees of support are sorted and $P\%$ of the values are dropped on both tails[6], conferring potential robustness to "outliers"; the $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$ is found as the `Mean` of the remaining degrees of support.

- `Generalized` (`Gen`) `Mean`:

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \left(\frac{1}{K}\sum_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)^{\alpha}\right)^{1/\alpha}$$

  comprises different means and functions as special cases.[7]

Finally, we consider also the `Probabilistic Product` (`PP`) aggregation [138], providing the maximum a-posteriori Bayes decision, based on the (unrealistic) assumptions that the classifiers use mutually independent subsets of features, and whose confidence measures yield the true posterior probability, that is $d_{k,i} = P(c_i|\hat{d}_k)$, on their respective feature subspaces. The combination formula is:

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \prod_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)/P(c_i)^{K-1},$$

where the prior probabilities $P(c_i)$ are estimated from training data.

2. *[CC] Trainable combiners*: here we will consider the (*i*) `Fuzzy`

[6] In the following, we have set $P\% = 20\%$ for our `Trimmed Mean` combiner.
[7] In the following, we have set $\alpha = \frac{1}{2}$ for our `Generalized Mean` combiner.

`Integral` approach (`FI`) and (*ii*) *trainable linear combinations*.

`FI` combiner searches for the maximal grade of agreement between the objective evidence, provided by the sorted classifier outputs for i^{th} class, and the expectation, namely the *fuzzy measure* values. More specifically, the `FI` is based on evaluating the support as:

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \max_{t=1}^{K} \left\{ \min \left\{ d_{k_t,i}(\boldsymbol{f}_T),\, g(t) \right\} \right\}.$$

In other terms, the vector $\boldsymbol{d}_i(\boldsymbol{f}_T)$ (*i.e.* the values of the support for c_i) is sorted in descending order and fused with the fuzzy measure for that class to get $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$. In the above equation, t^{th} element of the fuzzy measure for class c_i is denoted with $g(t)$ and its explicit formula is based on the accuracies of classifiers' pool and estimated through validation data. Therefore, for every test instance $\boldsymbol{f}_T$, L vectors of length K are evaluated, each corresponding to a class and containing values of the considered fuzzy measure.

Furthermore, we will consider the following *trainable linear combinations*:

- `K weights`:

  $$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \widetilde{\boldsymbol{w}}^T \boldsymbol{d}_i(\boldsymbol{f}_T),$$

  where $\widetilde{\boldsymbol{w}} \in [0,1]^{K\times 1}$ and $\widetilde{w}_k \triangleq \frac{(1/\epsilon_k)}{\sum_{t=1}^{K}(1/\epsilon_t)}$, being ϵ_k the (estimated) error-rate of k^{th} classifier [139].

- `KL weights`:

  $$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \boldsymbol{w}_i^T \boldsymbol{d}_i(\boldsymbol{f}_T),$$

  where $\boldsymbol{w}_i \triangleq (\boldsymbol{D}_i \boldsymbol{D}_i^T)^{-1} \boldsymbol{D}_i \boldsymbol{b}_i$ and $\boldsymbol{D}_i \in [0,1]^{K\times N}$ denotes the matrix obtained arranging all the i columns of the DP matrices belonging to the validation set, whereas $\boldsymbol{b}_i \in \{0,1\}^N$ whose n^{th}

entry equals 1 when the corresponding sample of the validation set belongs to c_i [140].

3. *[CI] Decision Templates (DT)*: the DT approach [130] stores the *most typical* DP for each class c_i (*i.e.* the DT of i^{th} class, denoted with $\bar{\boldsymbol{D}}_i$) and then compares it with the current DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ using a suitably chosen similarity measure $\mathcal{S}(\boldsymbol{D}(\boldsymbol{f}_T), \bar{\boldsymbol{D}}_i))$. The confidence for i^{th} class will be then:

$$\mu_i(\boldsymbol{D}(\boldsymbol{f}_T)) \triangleq \mathcal{S}(\boldsymbol{D}(\boldsymbol{f}_T), \bar{\boldsymbol{D}}_i)$$

when a new test instance $\boldsymbol{f}_T$ is submitted. Differently, during the training phase, the DT associated to i^{th} class $\bar{\boldsymbol{D}}_i$ is built as the average of the all the DP matrices within the validation set labeled with c_i.

In this study, we will employ three common similarity measures for the DT testing phase, based on the following distances: (a) squared Euclidean (`DT-SE`); (b) ℓ_1 norm after vectorization (`DT-L1`); (c) symmetric fuzzy-set originated (`DT-FSD`).

4. *[CI] Dempster-Shafer (DS) approach*: the present combiner takes its inspiration from the theory of evidence (viz. DS theory). Similarly to DT method, in the DS approach the DT matrices $\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_L$ are evaluated from the validation set. On the other hand, the similarity evaluation between each $\bar{\boldsymbol{D}}_i$ and the DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ is replaced by the following steps [130].

First, a $L \times K$ *proximity matrix* $\boldsymbol{\Phi}$ is built, whose $(i,k)^{th}$ entry is a normalized measure of distance[8] between the k^{th} rows of the i^{th}

[8] Although any distance could be employed, in our MCS we concentrate on ℓ_2 norm (`DS-L2`) for simplicity.

class DT $\bar{\boldsymbol{D}}_i$ and of the DP. In other words $\boldsymbol{\Phi}$ represents a similarity measure between the confidence vector of k^{th} classifier and its "typical profile" when c_i is the actual class. Secondly, by using $\boldsymbol{\Phi}$, for every class $c_i \in \Omega$ and for every classifier $k = 1, \ldots, K$, a *belief* degree $\beta_i\left(\boldsymbol{r}_k(\boldsymbol{f}_T)\right)$ is computed. Finally, the i^{th} degree of support $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T))$ is obtained as a normalized product of the belief degrees $\beta_i(\boldsymbol{r}_k(\boldsymbol{f}_T))$, with $k = 1, \ldots, K$.

3.3 Experimental Evaluation

In this section, we firstly provide a detailed description of the pre-processing operations (§3.3.1) carried out on the mobile multi-class datasets described in Sec. 2.5.1. We then report a systematic investigation of the effectiveness of these different pre-processing operations performed on data before actual classification (§3.3.2), so as to underline "best practices" by measuring their influence on the performance of all the considered classifiers/combiners. Finally, we report the performance of the proposed MCS and investigate its modularity in comparison to state-of-the-art classifiers devised for mobile TC (§3.3.3).

3.3.1 Dataset Pre-processing

In the successive analyses, the traces belonging (viz. the dataset corresponding) to the Android and iOS operating systems are investigated separately, in order to evaluate the detectability of mobile apps in a well established scenario (*i.e.* belonging to the same operating system / app store).

In the proposed MCS, after the burstification process, the network traffic is processed using the statistical features extraction block, both described in Sec. 3.2.1. We remark that the minimum SB length considered in this

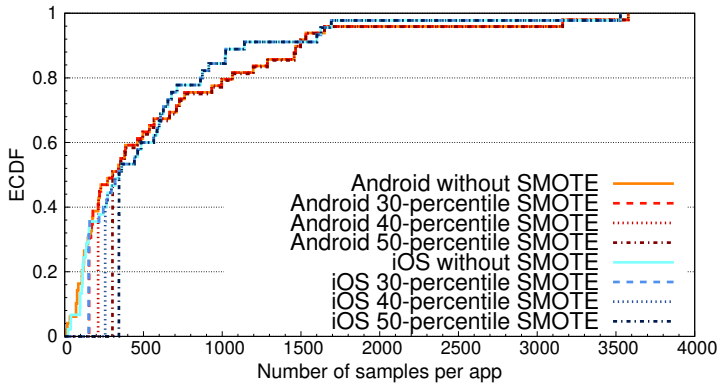


Figure 3.2: ECDFs of the number of samples per app for Android and iOS datasets before and after SMOTE application with different thresholds.

study is 7 (as suggested in [42]), since it is the shortest sequence of packets representing a meaningful data transfer which includes a TCP handshake and an HTTP/TLS request/response with corresponding ACKs. On the other hand, in our analysis, we do not restrict superiorly the length of the SB to be analyzed, since we did not consider—for reasons of computational complexity—the classification algorithms taking as input the varying raw vector of packets, referred to as "per-flow length classifiers" in [42]. We observe that, given the collection methodology of the considered traces introduced in Sec. 2.5.1, the SB definition is not prone to possible wrong-segmentation of the SBs within the same burst, according to the aggregation principle of the same destination IP address / port couple.

As evident from Tab. 2.4, the number of biflows for each app presents a severe imbalance that is especially true for the least observed ones. This is even exacerbated when considering the SB-segmentation as shown in Fig. 3.2 reporting the empirical CDF of the number of SBs per app for both Android and iOS datasets. Note that this distribution depends on the initial number

of SBs and therefore on the value of the BT whose impact is deepened in the next section. Generally, two different "philosophies" may be pursued for dealing with class imbalance. These mainly pertain to re-sampling (comprising oversampling and undersampling) methods [141] and cost-sensitive learning [63, 142] approaches (cf. Chapter 5).[9]

To deal with class imbalance problem, we apply an oversampling procedure to the datasets. More specifically, we employed the *Synthetic Minority Oversampling TEchnique* (SMOTE) [143] to the apps with a number of SBs less than the 30^{th} percentile of the distribution of the number of SBs per app to obtain a reasonable number of samples per app as shown in Fig. 3.2. SMOTE is one of the most popular approaches for data-based class-minority oversampling. Specifically, we adopted the filter implemented in the Weka environment [126] via `weka.filters.supervised.instance.SMOTE` Java class. We remark also that the results obtained with different percentages of SMOTE (*e.g.*, corresponding to the 40^{th} and 50^{th} percentiles as depicted in Fig. 3.2) have shown no discrepant relative performance among the classifiers and combiners (both hard and soft) considered in what follows, thus underlining the *stability* of the considered dataset. Specifically, after applying SMOTE, we obtained 30680 (resp. 25465) SBs composing our Android (resp. iOS) dataset, with the least populated classes composed by 155 (resp. 152) SBs. Finally, we underline that the present framework does not necessarily rely on SMOTE and such procedure can be safely removed from the pipeline in the case of a larger dataset. In the next section, we take into account the possibility to remove also *TCP retransmissions* and *zero-payload packets* and evaluate the impact of these further pre-processing steps on the "burstification" process.

[9]For an excellent introduction to different techniques which can be applied to imbalanced datasets, with specific focus on Internet TC, please refer to [141].

3.3.2 Impact of Dataset Pre-processing and Related Hints

Our first investigation on pre-processing steps applied to considered traces was aimed at assessing whether there is a substantial gain or, generically, a significant change in performance when cleaning traffic traces from *TCP retransmissions*. Results—not shown here for the sake of brevity—have underlined (almost) insensitivity of performance to the aforementioned operation, quantified in less than 0.2% change in accuracy for the best base classifier observed when considering a SB definition corresponding to $1s$ of BT. For this reason, in what follows, we have processed uncleaned (*i.e.* including TCP retransmissions) traffic traces, as the above step does not affect classification performance in a substantial way while adding unnecessary complexity to the proposed classification approach.

Then, two (coupled) useful investigations are pursued in what follows. First, we analyze the sensitivity of the classification performance to SB definition, focusing on the BT, to analyze whether and, in the affirmative case, to which degree, classifiers' performance are affected by this parameter. Indeed, previous studies have only provided results pertaining to empirically chosen values of the BT, corresponding to $1s$ [42, 43] and $4.5s$ [41], respectively. Hence, to provide a comprehensive BT analysis, we have employed the interval $[0.5, 5]$ seconds, with increments of $\Delta = 0.5s$, which includes both the aforementioned empirical choices.

Secondly, the aim is to investigate the potential gain achievable when removing zero-payload traffic. Indeed, a similar pre-processing step has been suggested in [144] for a WF task. Specifically, it has been advocated to remove packets sized 52 from features' evaluation, based on the intuition (confirmed by the appealing results in [144]) that packets of this length occur for all possible web pages, as these correspond to acknowledgments between

sender and receiver (*i.e.* TCP ACK packets with no payload). Thus, an evaluation of the features without packets sized 52 would allow to discard non-website-specific behaviors, which may be regarded as noise for the evaluation of the considered features. We argue that this may be also the case of mobile TC, as these packets correspond to a non-app-specific behavior. Accordingly, we pursue a similar (though more general, as some packets sized 52 could correspond to mobile data traffic exchange) approach, by *removing packets with zero-payload*.

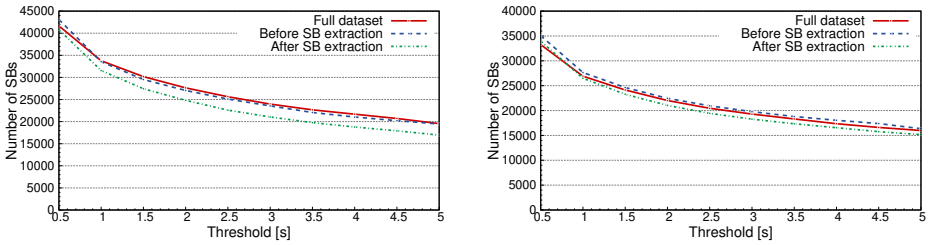
Since the two aforementioned processing steps are interdependent (*i.e.* the BT is influenced by the presence/absence of zero-payload packets), the following three configurations of the dataset have been considered, by varying the BT value:

(a) Original dataset (viz. no pre-processing).

(b) Dataset with zero-payload packets removed *after* the SB extraction.

(c) Dataset with zero-payload packets removed *before* the SB extraction.

Finally, SBs with a length less than *"Min Length"* packets (see Fig. 3.1) have been discarded in order to improve classification performance, as suggested in [42]. As mentioned before, in [42] a minimum flow length of 7 packets is considered as the optimal choice, since it represents the length of the shortest "complete" SB, that consists of a TCP handshake (three packets), an HTTP/TLS request/response pair (two packets) and the corresponding acknowledgments (two packets). Accordingly, we have made the same choice for both the cases (a) and (b). On the other hand, in case (c), we have selected a Min Length equal to 2, since in the latter case the SB extraction is performed on traffic traces whose zero-payload packets have been



(a) Android dataset.

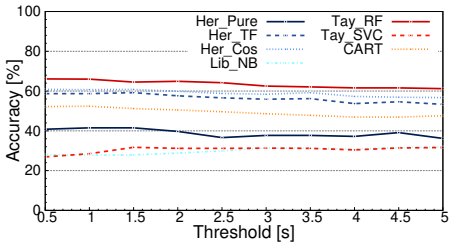
(b) iOS dataset.

Figure 3.3: Number of service bursts for different values of the BT considering Android (a) and iOS (b) datasets.

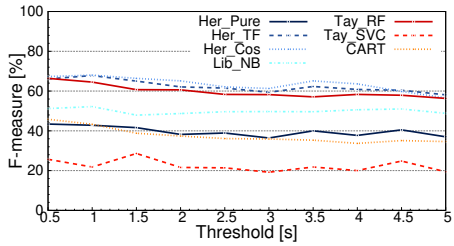
already removed (*i.e.* TCP handshake and acknowledgments belonging to the shortest "complete" SB).

Fig. 3.3 shows the number of SBs in these three datasets as a function of the BT. Intuitively, for all the datasets, the greater the BT, the lower the number of (resp. the longer the) SBs obtained. In detail, this number ranges from 16939 (resp. 15166) to 43089 (resp. 35064) for the Android (resp. iOS) dataset with zero-payload packets removal after (with a $5s$ BT) and before (with a $0.5s$ BT) the SB extraction, respectively. From the inspection of the figure, it can be noticed a "slope" change at BT equal to $1s$. It can be inferred that, when the BT is less than $1s$, the burstification process leads to an excessive fragmentation, and does not adequately capture the bursty nature of the considered mobile traffic. On the other hand, values higher than $1s$ may represent solutions which may lead to *merging* actually-distinct SBs, although in the range $(1, 5]$ seconds such "merging effect" seems not dramatic.

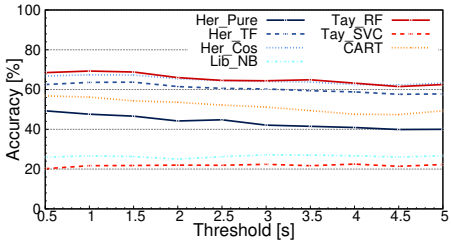
In Fig. 3.4 we show both the accuracy (left column) and the F-measure (right column) for all the classifiers described in Sec. 3.2.2 vs. the BT value, for the Android traces. More specifically, for each figure, cases (a)



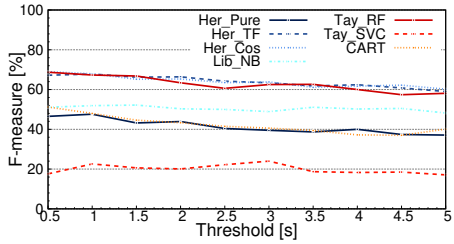
(a) Full dataset (Accuracy).



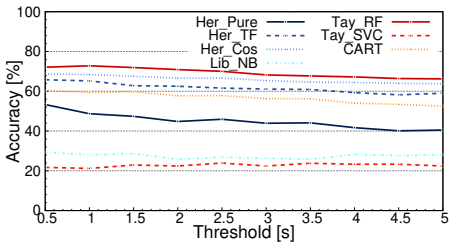
(b) Full dataset (F-measure).



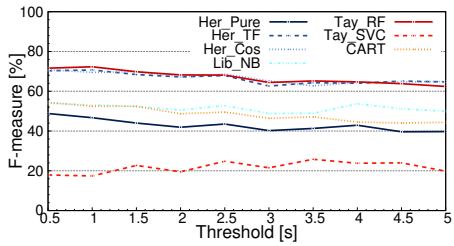
(c) Zero-payload packets removed *after* SB extraction (Accuracy).



(d) Zero-payload packets removed *after* SB extraction (F-measure).

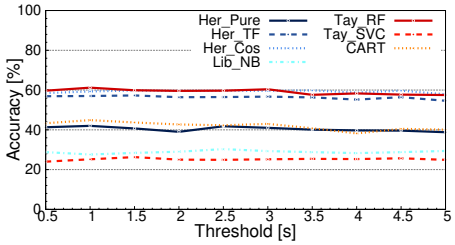


(e) Zero-payload packets removed *before* SB extraction (Accuracy).

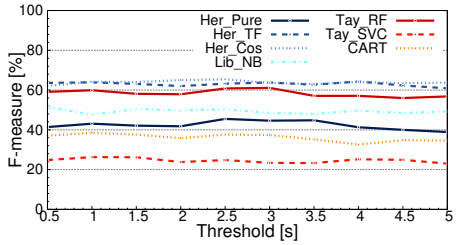


(f) Zero-payload packets removed *before* SB extraction (F-measure).

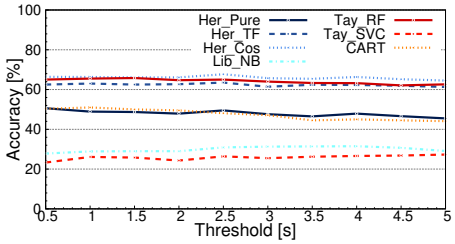
Figure 3.4: Accuracy (a, c, e) and F-measure (b, d, f) of the base (state-of-the-art) classifiers to varying the BT for the Android dataset.



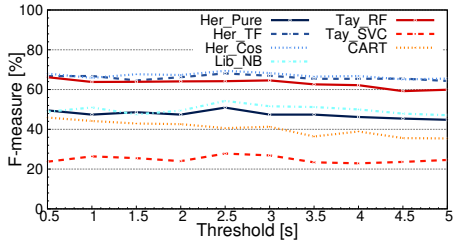
(a) Full dataset (Accuracy).



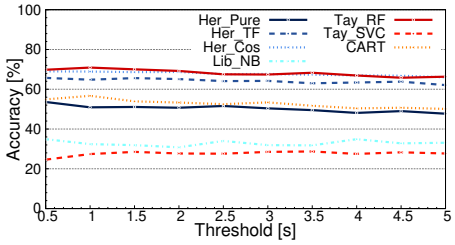
(b) Full dataset (F-measure).



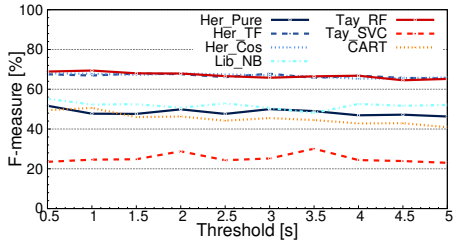
(c) Zero-payload packets removed *after* SB extraction (Accuracy).



(d) Zero-payload packets removed *after* SB extraction (F-measure).



(e) Zero-payload packets removed *before* SB extraction (Accuracy).



(f) Zero-payload packets removed *before* SB extraction (F-measure).

Figure 3.5: Accuracy (a, c, e) and F-measure (b, d, f) of the base (state-of-the-art) classifiers to varying the BT for the iOS dataset.

full dataset, (b) zero-payload packets removed *after* SB extraction, and (c) zero-payload packets removed *before* SB extraction are reported in top, middle, and bottom boxes, respectively. From the inspection of results, the highest performance is obtained with a threshold of $1s/1.5s$ in the case (c), *i.e.* with zero-payload packets removed before SB extraction. The optimal BT value found numerically also confirms the considerations on fragmentation/merging traffic effects arising from an inaccurate (viz. lower/higher) choice of the BT value, somewhat anticipated by the slope change phenomenon in Fig. 3.3. This trend can be observed for both Android and iOS traces shown in Fig. 3.5. However, in this latter case, base classifiers' performance exhibit a reduced dependence on the BT value, regardless of the removal of zero-payload packets. The results agree qualitatively with the considerations in [42, 43], thus underlining that $1s$ represents a good and stable choice for the BT. Nonetheless, in any case, automatic design and adaptability of this value would be desirable, being able to cope with networks experiencing different delay conditions.

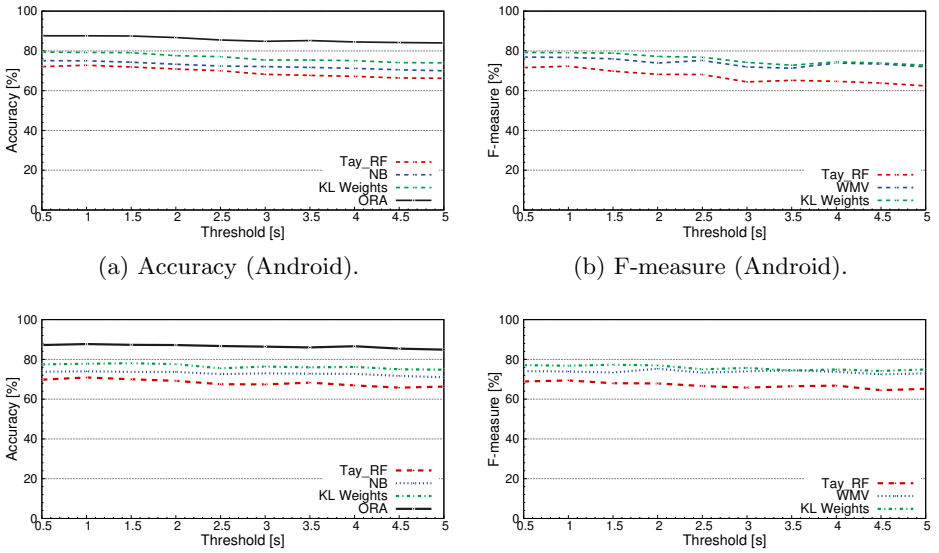
Similarly, it is evident that the removal of zero-payload packets *always* provides some gain in performance, which is independent on the specific BT considered, and whether such removal is performed after (b) or before (c) SB extraction. Nevertheless, an additional performance improvement is obtained when performing the removal before SB extraction (c). This may be explained as this filtering of "noisy packets" is also beneficial for a more effective SB segmentation. Indeed, taking into account a threshold value of $1s$, for the best base classifier (*i.e.* `Tay_RF`), the removal of zero-payload packets before and after SB partitioning produces an accuracy increment of +6.7% (resp. +9.7%) and +3.3% (resp. +4.3%) for Android (resp. iOS), respectively. Thus, as stated above, when this zero-payload cleansing is performed before, a further enhancement of +3.4% (resp. +5.4%) can be

obtained. Interestingly, F-measure increments—being in this case `Her_Cos` the best base classifier in terms of F-measure—are markedly smaller and substantial only for the removal of zero-payload packets before SB extraction: +1.5% (resp. +3.6%).

Additionally, it is apparent a weakly-decreasing trend for the best performing classifiers (namely `Tay_RF`, `Her_Cos`, and `Her_TF`) for increasing values of the BT. Similar trends can be observed for considered hard and soft combiners although less evident especially in the iOS case, since the influence of other base classifiers. The aforementioned behavior can be explained as larger values of the BT imply *longer* SBs, thus precluding a correct segmentation of the different actions associated to a certain app during time.

Therefore, since the removal of zero-payload packets before SB extraction seems an appealing pre-processing step over a wide range of BT values, in what follows we compare the performance of (*i*) the best base classifier (corresponding to `Tay_RF`, thus qualitatively agreeing with the results in [42, 43]), (*ii*) the best hard combiner (corresponding to either `NB` or `WMV` combiner, depending on the specific performance metric deemed relevant), and (*iii*) the best soft combiner (corresponding to the `KL weights`). The present investigation is conducted by measuring their accuracy and F-measure as a function of the BT over the same threshold range employed for Figs. 3.4 and 3.5. In Fig. 3.6, we report these results for Android and iOS traces. This allows investigating the general improvement provided by the present MCS system over the best base classifier, either considering hard or soft techniques, which is seen to be *almost independent* on the specific BT considered, with only Android traces showing little decreasing trend. For completeness, accuracy plots in Figs. 3.6a and 3.6c also report the performance of the `ORA` combiner[10], which highlights how the proposed approach

[10]Indeed, precision (and consequently F-measure) of the `ORA` cannot be evaluated since



(a) Accuracy (Android). (b) F-measure (Android).

(c) Accuracy (iOS). (d) F-measure (iOS).

Figure 3.6: Accuracy (a, c) and F-measure (b, d) of the best base classifier, hard and soft combiner versus the BT for the Android (a, b) and iOS (c, d) datasets. Performance refers to the dataset with zero-payload packets removed *before* the SB extraction.

"pushes" the performance toward the combining theoretical performance (*i.e.* upper-bound) for the considered pool.

3.3.3 Optimized Multi-Classification Results

Based on the previous considerations, in what follows we focus on case (c) (that is, removing zero-payload packets before burstification) and set the BT to $1s$, collectively representing the scenario with the highest performance observed. Then, we show results at a finer detail obtained by the application of the proposed MCS (see §3.2) to the aforementioned case. We remark that

its error patterns are not defined [76].

each considered analysis will employ a random training-validation-test set splitting (with corresponding percentages $50\% - 25\% - 25\%$, respectively).

Hard and Soft Combiners' Performance. First, in Tab. 3.3, we report the performance of all the base classifiers described in Sec. 3.2.2, in terms of the considered synthetic measures. Also, for completeness, we report the accuracy and recall achieved by the `ORA` in the rightmost column. From the inspection of results, it is apparent that `Tay_RF`, `Her_Cos`, and `Her_TF` achieve the highest performance with respect to the considered measures in the present setup, being still prone to classification errors. The quantitative scores seem, only at first glance, in contrast to those typically observed in Internet TC [76] and, more recently, to those achieved in the mobile context [42, 43]. However, in the former case, the classification problem is simplified by a homogeneous and less dynamic nature of the traffic being observed—typically coping with a lower number of classes to discriminate from—whereas in the latter case, it likely pertains to a non-exhaustive traffic collection procedure, being bot-generated and probably not capable of adequately "representing" all the "execution paths" of a generic app. Additionally, by looking at the `ORA` performance, the best accuracy (resp. recall) of the base classifiers can be improved by means of the proposed MCS up to 14.8% (resp. 19.5%) for Android and up to 16.8% (resp. 19.9%) for iOS, respectively. We notice that the upper-bound performance may be further improved by the adoption in the pool of other classifiers suitably-devised for mobile TC, confirming the appeal of the proposed MCS.

To this end, in Tab. 3.4 we show and compare the performance of the considered hard combiners. Results underline that `BKS` is able to provide the highest improvement with respect to the best base classifier (`Tay_RF`) in terms of overall accuracy. The same reasoning applies to `NB` for recall

Table 3.3: Performance (%) of base (state-of-the-art) classifiers considering Android (iOS) traffic. **Best base classifier** for each performance metric and dataset is highlighted in boldface.

Classifier	Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	ORA
Accuracy	48.7 (50.9)	65.2 (64.8)	68.4 (68.9)	28.0 (32.4)	**72.8 (70.9)**	21.2 (27.4)	59.4 (56.7)	87.6 (87.7)
Macro Precision	45.1 (47.2)	74.6 (70.0)	71.2 (69.3)	60.3 (60.7)	**74.7 (71.5)**	21.4 (30.0)	52.8 (50.9)	-
Macro Recall	54.8 (49.9)	58.4 (56.8)	63.5 **(62.3)**	36.0 (33.6)	**64.1 (62.3)**	9.89 (14.2)	51.4 (49.3)	83.6 (82.2)
Macro F-Measure	46.7 (47.7)	70.7 (66.9)	69.5 (67.8)	53.1 (52.3)	**72.3 (69.4)**	17.4 (24.6)	52.5 (50.6)	-

Table 3.4: Performance (%) of hard combiners considering Android (iOS) traffic. **Best hard combiner** for each performance metric and dataset is highlighted in boldface.

Combiner	MV	WMV	REC	NB	BKS	WER	ORA
Accuracy	72.2 (71.9)	72.8 (72.4)	73.8 (72.6)	**75.0** (74.0)	**75.0 (74.3)**	73.8 (71.9)	87.6 (87.7)
Macro Precision	79.3 **(76.9)**	**80.1 (76.9)**	78.7 (76.3)	75.8 (73.5)	77.4 (74.2)	75.6 (72.1)	-
Macro Recall	65.4 (63.4)	65.8 (63.9)	67.0 (64.2)	**70.7 (67.6)**	69.7 (67.2)	65.7 (63.5)	83.6 (82.2)
Macro F-Measure	76.1 (73.8)	**76.7 (73.9)**	76.1 (73.6)	74.7 (72.3)	75.7 (72.7)	73.4 (70.2)	-

measure, that also performs quite well in terms of accuracy. Differently, `MV` and `WMV` result appealing because of the remarkable improvement in terms of precision and F-measure over the best base classifier, being between +3.8% and +5.4%, respectively. Interestingly, `WMV` and `NB` represent the most appealing choices in terms of the set of performance metrics considered, almost collectively providing the highest performance. This is explained as they are less prone to over-fitting and have less training requirements, while also enjoying lower complexity with respect to `WER` and `BKS`. Remarkably, all the considered hard combiners (except for `MV`, when referring to the sole overall accuracy experienced with Android traffic) outperform the best base classifier in terms of all the considered performance metrics. This holds in both iOS and Android traffic.

A similar numerical comparison is shown in Tab. 3.5, where the performance of the three different groups of soft-combiners considered in this study are reported in separate sub-tables (*i.e.* CC non-trainable, CC trainable, and CI in sub-tables (*a-b*), (*c*), and (*d*), respectively), so as to (possibly) underline an interesting performance trend of a given group. For each group, `ORA` performance (as the rightmost column) is reported so as to highlight the corresponding improvement achievable.

By looking at their performance, it is apparent that a remarkable performance improvement can be achieved already with the sole use of CC non-trainable combiners. We remark that for these latter the availability of validation data is *not needed*. In fact, for the considered case, the `Mean`, the `Median`, the `Trimmed Mean`, and the `Generalized Mean` are able to improve `Tay_RF` performance in terms of all the reported synthetic measures. This holds in both iOS and Android traffic. On the other hand, the soft combination approaches provided by `PP`, `Maximum`, `Minimum`, `Harmonic Mean`, and `Geometric Mean` always lead to *unsatisfactory* performance when com-

Table 3.5: Performance (%) of different classes of soft combiners considering Android (iOS) traffic. **Best soft combiner** for each performance metric and dataset is highlighted in boldface.

(a) Class-conscious (CC) non-trainable combiners (1).

Combiner	Mean	Maximum	Minimum	Median	PP	ORA
Accuracy	75.3 (73.8)	61.5 (56.7)	51.8 (47.3)	73.7 (72.8)	52.1 (47.6)	87.7 (87.6)
Macro Precision	74.7 (71.8)	55.2 (50.2)	38.1 (35.3)	**79.8 (77.4)**	38.4 (35.6)	-
Macro Recall	**70.6** (67.5)	54.5 (50.5)	44.4 (40.0)	67.1 (64.2)	44.7 (40.3)	83.7 (82.3)
Macro F-Measure	73.8 (70.9)	55.0 (50.3)	39.3 (36.1)	**76.9 (74.3)**	39.5 (36.5)	-

(b) Class-conscious (CC) non-trainable combiners (2).

Combiner	Trim Mean	Harm Mean	Geom Mean	Gen Mean	ORA
Accuracy	75.6 **(74.4)**	51.8 (47.1)	52.3 (47.9)	**75.8 (74.4)**	87.7 (87.6)
Macro Precision	77.7 (75.1)	38.2 (35.1)	38.5 (35.6)	77.1 (74.9)	-
Macro Recall	70.4 **(67.7)**	44.4 (39.9)	44.5 (40.3)	70.5 (67.3)	83.7 (82.3)
Macro F-Measure	76.2 (73.5)	39.3 (35.9)	39.6 (36.5)	75.7 (73.2)	-

(c) Class-conscious (CC) trainable combiners.

Combiner	FI	K weights	KL weights	ORA
Accuracy	75.4 (73.5)	76.1 (74.2)	**79.2 (77.8)**	87.7 (87.6)
Macro Precision	77.0 (73.6)	76.3 (72.7)	**80.6 (78.2)**	-
Macro Recall	70.4 (67.2)	71.1 (67.8)	**73.6 (71.6)**	83.7 (82.3)
Macro F-Measure	75.6 (72.2)	75.2 (71.7)	**79.1 (76.8)**	-

(d) Class-indifferent (CI) combiners: Decision Templates (DT) and Dempster-Shafer (DS) approaches.

Combiner	DT-SE	DT-L1	DT-FSD	DS-L2	ORA
Accuracy	75.6 (72.6)	74.9 **(73.8)**	74.7 (72.8)	**75.7** (73.0)	87.7 (87.6)
Macro Precision	73.3 (69.1)	73.2 **(71.4)**	73.2 (69.9)	**73.5** (69.9)	-
Macro Recall	**72.6 (69.1)**	71.1 (68.4)	69.9 (66.8)	72.2 (69.0)	83.7 (82.3)
Macro F-Measure	73.1 (69.1)	72.8 **(70.7)**	72.5 (69.3)	**73.2** (69.7)	-

pared to the best base classifier (`Tay_RF`). This can be explained as these are more sensitive to soft-output misspecification of the classifiers in the pool.

Interestingly, a general remarkable improvement is achieved by the *whole group* of CC trainable combiners over `Tay_RF`. More specifically, though all the combiners within this group perform quite well, `KL weights` represents the most appealing choice in terms of all the measures considered and for traffic belonging to different OSs.

Furthermore, CI combiners are also able to improve, in most cases—except for a slight degradation of precision measure, see later discussion—the performance over the best base classifier, with `DT-L1` and `DS-L2` performing slightly better than others in the CC group. Still, the larger generalization capability of CI combiners does not pay back in terms of performance in comparison to CC trainable combiners. This may be attributed to an inadequate number of validation samples or to an over-fitting phenomenon. From a direct comparison of all the combiners belonging to all groups reported (both hard and soft), it is evident that `KL weights` represents the *best combiner* considered in this study for the present datasets. Finally, we underline that improved absolute performance measures may be achieved by the proposed MCS if additional (high performing and/or diverse) classifiers are developed to enlarge the considered pool.

Additionally, to summarize the improvement achieved by each group of hard/soft combining techniques, we have reported in Tab. 3.6 the *Maximum Improvement Over Best Classifier (MIOBC)* provided by each group for every performance measure. Referring to the group of hard combiners, it is apparent that such group is *always* able to provide an improvement, ranging from +2.2% (accuracy on Android traffic) to +6.6% (recall on Android traffic), by means of diversity principle, representing the milestone for adoption of classifier fusion techniques. However, as remarked before, different

Table 3.6: *Maximum Improvement Over Best Classifier* (MIOBC) of the F-measure (%) for each class of hard and soft combiners, considering Android (iOS) traffic. **Highest MIOBC** for each performance measure is highlighted in boldface. *Actual deterioration* is reported in italic.

Combiner	*Hard*	*CC non-trainable*	*CC trainable*	*CI*
Accuracy	+2.2 (+3.4)	+3.0 (+3.5)	**+6.4 (+6.9)**	+2.8 (+2.9)
Macro Precision	+5.4 (+5.4)	+5.1 (+5.9)	**+5.9 (+6.7)**	*-1.4 (-0,1)*
Macro Recall	+6.6 (+5.3)	+6.5 (+5.4)	**+9.5 (+9.3)**	+8.5 (+6.8)
Macro F-Measure	+4.4 (+4.5)	+4.6 (+4.9)	**+6.8 (+7.4)**	+0.8 (+1.3)

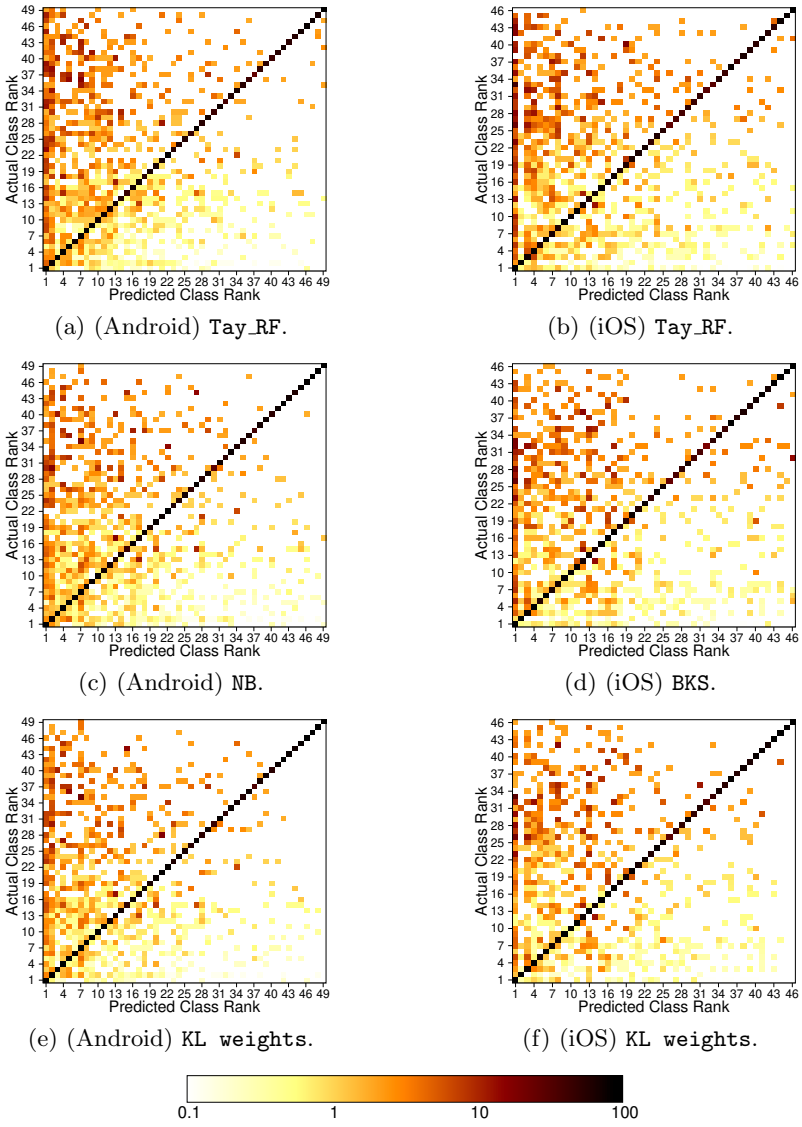
approaches (namely `MV`, `WMV`, `NB`, and `BKS`) result best according to different performance metrics. A similar reasoning applies to the group of CC non-trainable combiners (second column), where the improvement ranges from +3.0% (accuracy on Android traffic) to +6.5% (recall on Android traffic). Here, the improvement is qualitatively similar to hard combiners. However, CC non-trainable combiners, though requiring the availability of soft outputs from each classifier in the pool, do not require the availability of a validation set, which is instead required in the design of almost all the hard combiners here considered. This may be appealing in the case of scarcity of additional training (validation) data. On the other hand, the group of CC trainable combiners is able to provide the best improvement for each metric, up to +9.5% in terms of recall on Android traffic. This is not only the consequence of the presence of `KL weights` within the group, having the highest performance. Indeed, as observed earlier, all the combiners within the group are able to provide significant gains. Finally, CI combiners are able to collectively provide a performance improvement over almost all the considered metrics. The sole exception is represented by precision which is slightly degraded for all the combiners within this group, with a consequent gain reduction of the corresponding highest F-measure achieved by the CI

group.

Fine-grained Performance. We now compare the performance of the best classifier with the best hard and soft combiners at a finer detail, that is, by analyzing their confusion matrices, which allow to focus on misclassification patterns among apps (cf. §1.3.2). To this end, with reference to Android traffic, in Figs. 3.7a, 3.7c, and 3.7e, we show the confusion matrices of `Tay_RF`, `NB`, and `KL weights`, respectively. In addition, in Figs. 3.7b, 3.7d, and 3.7f, we report the same confusion matrices for iOS traffic. In this case, only the best-performing hard combiner differs, being in fact the `BKS`. It is worth noticing that similar qualitative trends have been observed for both Android and iOS traffic. Indeed, from the inspection of the results, it is revealed a homogeneously-reduced occurrence of misclassification patterns when employing a (good) combiner with respect to the best base classifier. Specifically, given the severe class-imbalance issue (cf. §3.3.1) suffered by both datasets[11], the confusion matrix of the (best) base classifier reveals a bias toward the most-populated classes that is considerably mitigated when considering our MCS. This is even more evident when a soft combiner—being for both OSs the `KL weights`—is employed.

Classifiers' Pool Selection. In Tabs. 3.7, 3.8, and 3.9 we delve into how classifiers' subset selection affects performance, focusing on the F-measure. The intent is investigating possible performance gain of the considered combiners (grouped as done in Sec. 3.2.3) and computational complexity reduction, by discarding *non-informative* classifiers from the pool. Since the number of different subsets is combinatorial and having available different optimization criteria (*i.e.* considered combiners), it is impractical evaluat-

[11] This actually represents a realistic and challenging scenario.



(a) (Android) `Tay_RF`.

(b) (iOS) `Tay_RF`.

(c) (Android) `NB`.

(d) (iOS) `BKS`.

(e) (Android) `KL weights`.

(f) (iOS) `KL weights`.

(g) Log-scale (%).

Figure 3.7: Confusion matrices of the best base classifier (a, b), hard combiner (c, d), soft combiner (e, f) for Android (a, c, e) and iOS (b, d, f). Labels refer to apps in Tab. 2.4, but are ranked according to decreasing abundance of samples. Logarithmic scale (g) is used to evidence small errors.

ing the performance for all the possible combinations. Hence, we adopt an heuristic approach informed by the *diversity* of classification methods and iteratively removing the *worst performing* classifier.

Referring to the hard combiners (cf. Tab. 3.7), we can make several observations. The best overall F-measure performance is achieved by `MV` and `REC` on iOS and Android traffic, respectively. The appeal of this result is that these combiners have low requirements both in terms of training samples and operational (viz. testing phase) complexity. Additionally, it is apparent that the hard combiners requiring the least parameters to be trained (*i.e.* `MV`, `WMV`, `REC`, and `NB`) all benefit from the selection of a subset of classifiers within the pool. Interestingly, they all achieve their *maximum per combiner* when only `Her_Cos`, `Lib_NB`, and `Tay_RF` are employed. This may be attributed at the higher diversity provided by these three different base classifiers. On the other hand, `WER` also presents improved performance with a different selection of the subset of classifiers. Specifically, it requires a larger subset for Android traffic (*i.e.* all three `Her` variants, `Lib_NB`, and `Tay_RF`), whereas only `Her_Cos` and `Tay_RF` are needed in the pool to achieve its highest performance over iOS traffic. Finally, it is apparent that `BKS` does not benefit from the same subset selection as `MV`, `WMV`, `REC`, and `NB`. Therefore, we argue that this may be attributed to over-fitting issues (*i.e.* unnecessarily modeled correlation between diverse base classifiers).

Then, with reference to CC non-trainable combiners (cf. Tabs. 3.8a and 3.8b), we first observe that `PP`, `Maximum`, `Minimum`, `Harmonic Mean`, and `Geometric Mean` combiners have a dramatic improvement of F-measure performance when considering small subsets of the classifiers' pool. Similarly, the `Mean`, `Median`, `Trimmed Mean`, and `Generalized Mean` are able to improve (almost always) their performance when considering the smallest pool composed by `Her_Cos` and `Tay_RF`. However, their performance im-

provement is less steep. This trend may be explained as CC non-trainable combiners are more prone to be biased from wrong classifiers in the pool, due to the lack of high-level (viz. validation-based) training. Nevertheless, the latter sub-group possesses an intrinsic robustness to having outliers in the pool, due to their peculiar combination functions.

On the other hand, by observing the performance of CC trainable combiners (cf. Tab. 3.8c), it is apparent that `KL weights` benefits from a judicious use of the subset. This allows reducing the number of parameters to be trained, especially those related to weak classifiers, and thus avoids slight over-fitting. A different trend is instead observed for `K weights`, being similar to that observed for CC non-trainable combiners. The reason is that the linear (separating) vector employed is based on the assumption that each soft-output well-matches the actual one, except for some estimation noise [139]. Therefore, this approach is potentially sensitive to erroneous (*i.e.* providing incoherent soft-outputs) base classifiers. A somewhat similar behavior as `BKS` is observed for `Fuzzy Integral`, which does not benefit from subset selection. This may be attributed to the fact that the proposed fuzzy-based fusion design is resistant to classifiers' uncertainty.

A less evident trend can be drawn for CI combiners (cf. Tab. 3.9). Nonetheless, it can be concluded how all the proposed approaches achieve the highest F-measure with the group considered by `Her_Cos`, `Lib_NB`, and `Tay_RF`, with the sole exception of `DT-FSD` in Android traffic, where the smallest group composed by `Her_Cos` and `Tay_RF` should be employed to reach the highest performance.

A summarizing comparison, reporting the MIOBC (in terms of F-measure) for each group of combiners, is shown in Tab. 3.10, exploring the same subset choices previously considered. From its inspection, it is apparent how overall improved performance with respect to considering the whole

pool of classifiers can be obtained on Android traffic (*i.e.* +0.5% employing `Geom Mean` on `Her_Cos` and `Tay_RF` against `KL weights` on the whole pool) or the same results with less training requirements (*i.e.* using a CC non-trainable combiner rather than a CC trainable one) in iOS traffic. It is worth highlighting that, the modularity of the considered MCS allows its virtual application to other suitably-devised classifiers for further performance enhancement (as confirmed also by `ORA` performance), as DL-based (multimodal) base classifiers discussed in Chapter 5. Moreover, the MCS could be integrated with other advanced classification structures like hierarchical ones, described in the next Chapter.

Table 3.7: F-measure (%) of hard combiners as function of the pool of selected classifiers considering Android (iOS) traffic. Highlighted values: **maximum per pool**, *maximum per combiner*, overall maximum.

Pool of classifiers							**Combiners**					
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	MV	WMV	REC	NB	BKS	WER
✓	✓	✓	✓	✓	✓	✓	76.1 (73.8)	**76.7 (73.9)**	76.1 (73.6)	74.7 (72.3)	*75.7 (72.7)*	73.4 (70.2)
✓	✓	✓	✓	✓		✓	75.7 (72.8)	**76.4 (73.3)**	75.5 (72.7)	74.7 (71.8)	74.8 (70.4)	73.3 (69.8)
✓	✓	✓	✓	✓			73.1 (69.7)	**73.8** (70.1)	73.0 (69.9)	72.7 **(70.9)**	71.4 (68.3)	***73.8*** (70.6)
	✓	✓	✓	✓			76.3 (73.2)	**76.7** (73.8)	76.3 **(73.9)**	74.8 (72.7)	73.0 (69.9)	73.6 (70.4)
		✓	✓	✓			*77.5* ***(77.6)***	*77.2 (76.0)*	***77.7*** *(77.0)*	*75.7 (75.7)*	70.0 (67.8)	72.9 (69.7)
		✓		✓			75.1 (73.2)	75.1 (73.2)	**75.4 (73.6)**	74.4 (74.0)	71.5 (69.0)	73.1 *(70.9)*

Table 3.8: F-measure (%) of soft combiners as function of the pool of selected classifiers considering Android (iOS) traffic. Highlighted values: **maximum per pool**, *maximum per combiner*, overall maximum.

(a) Class-conscious (CC) non-trainable combiners (1).

Pool of classifiers							Combiners				
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Mean	Maximum	Minimum	Median	PP
✓	✓	✓	✓	✓	✓	✓	73.8 (70.9)	55.0 (50.3)	39.3 (36.1)	**76.9 (74.3)**	39.5 (36.5)
✓	✓	✓	✓	✓		✓	73.2 (70.4)	55.2 (50.7)	39.6 (36.1)	74.6 (72.2)	39.6 (36.3)
✓	✓	✓	✓	✓			70.4 (67.7)	62.3 (60.9)	52.0 (51.3)	71.5 **(69.1)**	55.6 (55.1)
	✓	✓	✓	✓			75.7 (73.1)	71.6 (70.0)	64.0 (61.6)	76.3 (73.7)	67.5 (66.3)
		✓	✓	✓			74.8 (73.9)	71.1 (70.2)	64.7 (63.0)	76.6 **(75.3)**	67.9 (66.3)
		✓		✓			*77.5 (74.1)*	*76.6 (72.8)*	*79.0 (75.2)*	*77.5 (74.1)*	*75.8 (73.1)*

(b) Class-conscious (CC) non-trainable combiners (2).

Pool of classifiers							Combiners			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Trim Mean	Harm Mean	Geom Mean	Gen Mean
✓	✓	✓	✓	✓	✓	✓	76.2 (73.5)	39.3 (35.9)	39.6 (36.5)	75.7 (73.2)
✓	✓	✓	✓	✓		✓	**75.5 (72.7)**	39.5 (36.2)	39.8 (36.1)	74.6 (72.2)
✓	✓	✓	✓	✓			**71.9** (69.0)	52.3 (51.8)	58.7 (56.5)	71.3 **(69.1)**
	✓	✓	✓	✓			76.3 (73.7)	65.2 (62.1)	72.1 (68.2)	**76.6 (74.4)**
		✓	✓	✓			**76.7** ***(75.3)***	65.1 (63.5)	71.5 (67.8)	76.3 *(75.7)*
		✓		✓			*77.5* (74.1)	*79.2 (75.7)*	***79.6 (76.8)***	*78.8* (75.3)

(c) Class-conscious (CC) trainable combiners.

Pool of classifiers							Combiners		
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Fuzzy Integral	K weights	KL weights
✓	✓	✓	✓	✓	✓	✓	*75.6 (72.2)*	75.2 (71.7)	**79.1** ***(76.8)***
✓	✓	✓	✓	✓		✓	70.5 (70.4)	72.1 (69.2)	***79.4*** ***(76.8)***
✓	✓	✓	✓	✓			72.0 (69.4)	73.5 (69.8)	**79.3 (76.4)**
	✓	✓	✓	✓			72.3 (70.7)	74.9 (72.9)	***79.4*** **(76.5)**
		✓	✓	✓			72.9 (71.0)	76.3 *(73.6)*	**78.8 (76.2)**
		✓		✓			71.6 (69.3)	*76.7* (73.1)	**78.2 (75.1)**

Table 3.9: F-measure (%) of soft combiners as function of the pool of selected classifiers considering Android (iOS) traffic. Highlighted values: **maximum per pool**, *maximum per combiner*, overall maximum. Class-Indifferent (CI): Decision Templates (DT) and Dempster-Shafer (DS) approaches.

Pool of classifiers							**Combiners**			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	DT-SE	DT-L1	DT-FSD	DS-L2
✓	✓	✓	✓	✓	✓	✓	73.1 (69.1)	72.8 **(70.7)**	72.5 (69.3)	**73.2** (69.7)
✓	✓	✓	✓	✓		✓	73.1 (69.1)	72.8 **(70.8)**	72.4 (69.3)	**73.2** (69.5)
✓	✓	✓	✓	✓			**70.5** (67.7)	68.8 (66.9)	68.1 (65.5)	70.2 **(67.8)**
	✓	✓	✓	✓			**73.1 (70.6)**	72.5 (69.3)	72.2 (70.1)	72.8 (70.2)
		✓	✓	✓			*73.7 (72.0)*	***74.1*** *(71.5)*	73.3 ***(72.2)***	*73.9 (71.7)*
		✓		✓			73.3 **(70.3)**	**73.9** (70.2)	*73.7* (69.5)	73.2 (70.1)

Table 3.10: *Maximum Improvement Over Best Classifier* (MIOBC) of the F-measure (%), as function of the pool of selected classifiers, for each class of hard and soft combiners, considering Android (iOS) traffic. *Actual deterioration* is reported in italic.

Pool of classifiers							**Combiners**			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Hard	CC non-trainable	CC trainable	CI
✓	✓	✓	✓	✓	✓	✓	+4.4 (+4.5)	+4.6 (+4.9)	+6.8 (+7.4)	+0.9 (+1.3)
✓	✓	✓	✓	✓		✓	+4.1 (+3.9)	+3.2 (+3.3)	+7.1 (+7.4)	+0.9 (+1.4)
✓	✓	✓	✓	✓			+1.5 (+1.5)	*-0.4 (-0.3)*	+7.0 (+7.0)	*-1.8 (-1.6)*
	✓	✓	✓	✓			+4.4 (+4.5)	+4.3 (+5.0)	+7.1 (+7.1)	+0.8 (+1.2)
		✓	✓	✓			+5.4 (+8.2)	+4.4 (+6.0)	+6.5 (+6.8)	+1.8 (+2.8)
		✓		✓			+3.1 (+4.2)	+7.3 (+7.4)	+5.9 (+5.7)	+1.6 (+0.9)

Chapter 4

Hierarchical Classification Framework

In line with the growing criticality of the activities users perform online, privacy represents one of the key concerns regarding traffic analysis. In Chapter 1, we have remarked how TC is tightly-coupled to the privacy aspect, with a lot of effort spent in developing privacy-preserving solutions and, on the other hand, understanding to what extent these solutions are effective (*e.g.*, against classification). In particular, preserving the anonymity of users is an aspect that has gathered the attention of the research community, that over the last years has put the effort in designing and developing tools able to achieve privacy at varying degrees. As a result, at present a number of *Anonymity Tools (ATs)* are freely available, able to hide—besides the content of the communication itself, via encryption—the identity of the parties (*i.e.* source and destination) involved in the communication and in many cases they are even capable of hiding the users' identity to the final destination (*i.e.* the web-server). The most popular ATs developed in recent years are The Onion Router (TOR) [70], the Invisible Internet Project (I2P) [124], and JonDonym (formerly known as Java Anon Proxy or Web-Mix) [125].

These tools allow users to preserve their anonymity—*e.g.*, encrypting data multiple times and routing it through multiple stations—providing each with just a piece of the information. From the user viewpoint, the ATs allow browsing and running applications also circumventing restrictions enforced at either providers or governmental level, keeping their identity and location secret to any intermediary entity observing the traffic. Accordingly, tracing users and their activities across these networks is a complex task.

In recent years, several studies have investigated ATs from different perspectives including: design improvement, AT delay and performance analysis, feasibility of effective attacks to ATs, users' behavior profiling and identity disclosure risk, and censoring policies enforced for ATs [145]. Among these crucial aspects, a cardinal issue is *understanding to what extent encrypted ATs' traffic can be classified*. More specifically, it is interesting to ascertain to which degree an external observer can recognize an AT and how fine would be the fingerprinting granularity achievable, that is, whether traffic types and/or services/applications hidden into them could be inferred.

On the one hand, investigating TC of ATs is useful to designers, as it puts their effectiveness to the test, reveals shortcomings, and leads the way to robustify them. On the other hand, these studies are of interest to providers as well as governmental entities, providing knowledge for example, to enforce informed engineering policies or to prevent users performing unwanted actions. Hence, classifying ATs' traffic is a particularly appealing and challenging research field whose state-of-art solutions leave room for improvement.

In this regard, the classification of ATs' traffic represents an open and challenging task in the realm of encrypted TC tackled in this Thesis. Indeed, as already discussed in Sec. 1.3.4, given this encrypted nature, ML

classifiers represent the natural enabler of its classification, overcoming the shortcomings due to the application of traditional solutions. On the top of that, hierarchical classification represents a perfect match for TC of ATs, as (i) it allows fine-grained tuning and design, potentially leading to classification performance gains; (ii) it also brings a number of *"practical" benefits by design*, at cost of moderate complexity increase. For example, re-training does not involve all the nodes in the hierarchy when new applications leveraging anonymity networks are released. Also, distributed deployment of TC tasks, thanks to the modularity of the framework, is enabled in the network, thus hierarchical classification could be achieved through chaining of virtualized network functions, each associated to a classifier. Albeit these benefits are granted by the hierarchical approach itself, research efforts are needed to deepen the aspects of design optimization (to obtain enhanced performance) and fine-grain evaluation to delve into privacy-level assessment of ATs.

In view of these considerations, the present chapter proposes a *general Hierarchical Classification (HC) framework* (see Fig. 4.1) for encrypted TC, carrying out a detailed study on encrypted ATs' traffic. This framework represents another "structural" improvement over standard ML-based classifiers, complementary to the MC architecture described in Chapter 3. We deal with its design, implementation, optimization, and evaluation. In detail, our analysis has a *three-fold* significance: (i) optimizes a pool of flat classifiers to study minutely the number and nature of relevant features needed for an accurate classification, (ii) investigates the unexplored adoption of hierarchical approaches to TC of ATs as opposed to optimized flat learning approaches, and (iii) overcomes the limitations of earlier hierarchical proposals dealing with standard (viz. non-encrypted/non-anonymous) TC. Such HC framework *uniquely suits* to encrypted traffic and *naturally*

allows for both coarsening and narrowing of classification results.

Specifically, the hierarchical approach resorts to a structure of (potentially different) classifiers arranged in a *tree* fashion, each specialized in labeling a subset of classes. Such framework exploits the "divide-et-impera" principle, enabling the partition of workload among several classifiers, almost-naturally enabling a distributed implementation. Hence, the obtained HC structure grants scalability, enables both per-node tuning and performance analysis, and supports roll-up and drill-down operations of the results, which are provided at different levels of detail. Per-node performance figures also allow to accurately evaluate per-node behaviors and to identify potential causes of performance degradation, thus proving useful in guiding feedback-driven design improvement. Besides, HC design supports incremental-updates, for example only a minor additional training is required when a new part of the application traffic is needed, instead of re-training the whole system. Finally, the proposed architecture allows progressive censoring of "unsure" instances, implementing a per-classifier *reject option* (cf. §1.3.2) via a set of independently-tunable thresholds.

The key issue researchers encounter in the collection of AT traffic datasets and the resulting lack of data publicly available in this field—which prevents experiments repeatability and, as a matter of fact, precludes unanimous and shared conclusions, as discussed in Chapter 2—is herein overcome by leveraging the recently released Anon17 dataset (cf. §2.5.2 and [44] for details). It constitutes an important shared workbench for research studies on anonymity and consists of a collection of traces gathered by different ATs, at different granularities: *anonymous network*, *traffic type*, and *application*, that represent also the levels at which we carry out our analysis. Although our HC strategy is encouraged by the nature of the classification problem defined on this dataset, the underlying HC principle generally

suits to the varying degrees of privacy ensured by a certain AT, including its identification within "normal" traffic.

The results show that the proposed HC approach proves to be a very good fit to the problem of classification of ATs' traffic. It is able to discern among ATs looking at their encrypted traffic, also when considering only the features extracted from the first K packets of each flow (*i.e.* implementing *early TC*, see §1.3.1). To prove its effectiveness, we compare our HC methodology with various flat approaches, considering five ML-based classifiers: three of them are based on the *Bayesian approach* (*i.e.* Naïve Bayes, Multinomial Naïve Bayes, and Bayesian Networks), whereas the other two on the well-known *decision trees* (*i.e.* C4.5 and Random Forest). In this way, we can not only compare different approaches (both among themselves and with our HC), but we can also investigate the significant features required for an accurate classification, as well as the need to reach conclusions not coupled to a specific classifier. Indeed, we will show that the hierarchical framework, in addition to its unique advantages, also provides performance gains at the application level. This result holds both when considering classic performance indexes at macroscopic level (typically adopted in the TC literature) and by accurately breaking it down and evaluating it along multiple facets (*i.e.* by introducing metrics able to capture error severity).

The rest of the chapter is organized as follows. We recap the most related works in Sec. 4.1, considering those leveraging hierarchical approaches or tackling TC of ATs. Successively, Sec. 4.2 describes the considered hierarchical TC framework of ATs. The experimental environment and the corresponding classification results (encompassing flat and hierarchical classification performance along with summarizing considerations) are reported in Secs. 4.3 and 4.4, respectively.

4.1 Related Works

Up to our knowledge, there are no studies focused on classification of different anonymity services at various levels of granularity via *hierarchical learning*. Accordingly, this section first deepens the few works tackling standard TC (*i.e.* not focusing on traffic generated by ATs) via a hierarchical approach (see Tabs. 1.2 and 4.1). Then, we integrate this study with a brief review of the literature tackling TC of ATs via a "flat" (viz. non-hierarchical) approach, including the conceptually-related WF problem.

4.1.1 Hierarchical Traffic Classification

Table 4.1 summarizes the details of previous works leveraging hierarchical frameworks for TC. As already mentioned, none of them have dealt with anonymous TC, whereas all (with the sole exception of the approach proposed in [82, 83], whose detailed description is not provided) are able to operate under the encryption assumption (*i.e.* they are all based on ML/DL classifiers not relying on cleartext payload-based features). We detail each work in the following.

A first approach to hierarchical TC is described in [75], where a three-level system for Peer-to-Peer (P2P) biflow-based traffic categorization is proposed to discern among 11 P2P and 5 non-P2P apps. The first-level classifier adopts a Support Vector Machine (SVM) for P2P/non-P2P recognition, whereas multi-class SVMs based on Support Vector Data Descriptions are employed at the other two levels for P2P-type (*i.e.* file-sharing, messenger, and TV) and P2P plus non-P2P app TC, respectively. Results (pertaining to each single classifier in the hierarchy considered *separately*) show that the proposed approach achieves precision and recall $\geq 95\%$ in P2P/non-P2P recognition and $\geq 93\%$ in P2P-type classification, while a re-

Table 4.1: Summary of previous works (by year) tackling TC via hierarchical approaches.

Paper	ET	AT	TO	#Levels	Classes	Input Data	Classifier	TC Performance
Yu *et al.* [75]	●	○	BF	3	L1: P2P/nonP2P (2) L2: P2P type (3) L3: Application (16)	Protocol; duration; BC; PC; statistics of PS, WS, & IAT	SVM	$\geq 71\%$ rec.
Grimaudo *et al.* [77]	●	○	F	4	L1: Known/unseen (2) L2: Protocol (3) L3: Application (14) L4: Video Stream (4)	≤ 35 mRMR-selected features from 200 total	NB, BKE, RB, DT, NN, SVM, K-NN	$\approx 95\%$ F-meas.
Yoon *et al.* [82, 83]	○	○	BF	N/A	N/A	Header, payload, or statistical features	N/A	GT not available
Shbair *et al.* [85]	●	○	BF	2	L1: Provider (68) L2: Service (≈ 100)	Statistics of PC, PS, & IAT	C4.5, RF	93.1% acc.
Dong *et al.* [88]	●	○	BF	2	L1: (A)symmetric (2) L2: Video App (6)	Statistics of PC, PS, & IAT	K-NN	$\geq 97\%$ acc.
Chen *et al.* [96]	●	○	P	2	L1: Service (≤ 24) L2: Application (N/A)	IP packet [39×39 B]	2D-CNN	$> 85\%$ acc.
Our HC approach	●	●	*F*	3	*L1: Anon Network (3)* *L2: Traffic Type (7)* *L3: Application (21)*	81 *per-flow features* *First K PS*	*NB, MNB, BN, C4.5, RF*	+4.5% *F-meas. w.r.t. best flat*

Encrypted Traffic (ET). **Anonymous Traffic (AT)**.
Traffic Object (TO): biflow (BF), flow (F), packet (P)
Input Data: byte count (BC), inter-arrival times (IAT), packet count (PC), packet sizes (PS), TCP-window sizes (WS), minimum Redundancy Maximum Relevance (mRMR).
Classifier: Bayesian Kernel Estimation (BKE), Bayesian Network (BN), Convolutional Neural Network (CNN), Decision Tree (DT), K-Nearest Neighbors (K-NN), Multinomial Naïve Bayes (MNB), Naïve Bayes (NB), Neural Network (NN), Random Forest (RF), Rule Based (RB), Support Vector Machine (SVM).
*N/A: information not available in the related manuscript.

call drop down to 71% is observed at the last level. Similarly, Grimaudo *et al.* [77] propose the adoption of a hierarchical classifier to allow Internet TC (into ≥ 20 fine-grained classes), showing a comparison with flat-learning results. The proposed tree-structured (four-level) taxonomy introduces also a first level which identifies known/unseen traffic. Both per-classifier feature selection (from a set of 200, as proposed by past literature) and coarse-grained optimization with respect to different ML classifiers (see Tab. 4.1) are considered. Results show that the proposed hierarchical system outperforms off-the-shelf flat classification, with an average F-measure/recall $\approx 95\%$ for the most popular traffic classes.

In [82] a novel *multilateral* (viz. multi-view) and hierarchical approach for Internet TC is proposed, further refined into the *FORMULA* framework [83], operating on biflow-segmented traffic. Specifically, the devised approach relies on a taxonomy that provides multilateral identification based on four different classification criteria (*i.e.* service, application, protocol, and function) via a hierarchical structure—whose working principle is however not detailed—supporting roll-up and drill-down operations on the classification results. Unfortunately, the collected traffic lacks a ground truth, precluding a verification of the reported results.

Recently, in [85] a two-level hierarchical classification framework is presented to identify the services running within HTTPS connections leveraging a set of features robust to alteration (*e.g.*, statistics of inter-arrival times and packet/payload sizes). The proposed evaluation method, based on real traffic traces, achieves a recall within $[95, 100]\%$ in 50 out of the 68 HTTPS services considered. Dong *et al.* [88] propose a fine-grained scheme for hierarchical TC of video traffic based on clustering. A hierarchical version of K-NN is fed with the most discriminative statistical features selected among 40 extracted from downstream/upstream data, ranked by the information

gain ratio. The dataset, including six types of *symmetric* and *asymmetric* traffic, is collected on the campus network of Nanjing University. Experimental results report $\geq$ 97% F-measure in discriminating among all the considered video traffic and superior performance with respect to existing alternatives, while providing only a slight time-complexity increase.

Finally, in [96] a DL-based scheme for encrypted packet classification is proposed. Hierarchical structure is composed of two levels, service level and application level. The former is potentially able to discriminate between 24 classes (*e.g.*, video streaming, online chatting, online gaming, etc.), whereas no details are given for the application-level classifiers. Authors perform the experimental evaluation using a bi-dimensional Convolutional Neural Network and considering only 2 services (*i.e.* data-chat and video) encompassing 5 and 1 applications, respectively. The best performance, shown only for the single classifiers (and not for the whole hierarchy), is obtained by the data-chat classifier fed with a matrix of 39×39 B packet-data.

4.1.2 Flat Traffic Classification of Anonymity Tools

The analysis of ATs has been initially carried in *private networks*, that is with the aim of discriminating between HTTPS and Tor traffic [146]. In detail, by leveraging a dataset made of (*i*) regular HTTPS traffic, (*ii*) HTTP, and (*iii*) HTTPS over a private Tor network, authors show that HTTP / HTTPS traffic over Tor can be detected with $\geq$ 93% accuracy, employing RF, C4.5, and AdaBoost classifiers.

On the other hand, a few works focus on TC analyzing *real traffic* from anonymity networks, as most of the "experimental" literature explores anonymous WF, whose aim is to identify a web-page accessed by a client of encrypted and anonymized connections by observing patterns of data flows (*e.g.*, packet size and direction). Herrmann *et al.* [65] propose a MNB that

relies on the normalized frequency distribution of IP packet sizes to tackle the WF problem in the context of different privacy-enhancing technologies (in a closed-world scenario) including Tor and JonDonym. Although Tor and JonDonym guarantee a better protection than other privacy-enhancing technologies (*i.e.* OpenSSL and OpenVPN), they prove to be not perfect (3% and 20% average accuracy, respectively). In [144] the same problem is tackled via a SVC, obtaining a gain of detection rate over [65] from 3% to 55% (resp. from 20% to 80%) in Tor (resp. JonDonym) network. On the other hand, in an open-world case, the detection rate drops with a maximum of 73% and 0.05% false-positive rate. More recently, in [147] a WF approach aimed to overcome limitation of previously-devised alternatives is proposed and tested on a huge real-world representative dataset, exploring the limits of WF at Internet scale. Specifically, these are highlighted by a precision/recall drop with the size of the background sites which the monitored pages need to be distinguished from. Finally, we mention that the adoption of DL to WF is also a currently-investigated topic. A novel DL-based method to deanonymize Tor traffic is proposed in [25] and tested on a dataset made of $\geq 3 \cdot 10^6$ network traces, with the best-performing DL model being +2% accurate than state-of-the-art attacks. In [26] a WF attack against Tor is developed, leveraging a Convolutional Neural Network and evaluated against state-of-the-art defenses (*i.e.* WTF-PAD and Walkie-Talkie). Results report an accuracy $> 98\%$ on undefended Tor traffic, while reaching $> 90\%$ accuracy (resp. 49.7%) when WTF-PAD (resp. Walkie-Talkie) is employed, with the attack remaining effective also in an open-world setting.

Moving to pure TC of ATs based on real-data, He *et al.* [148] devise an approach based on Hidden Markov Models (HMMs) to classify four categories of Tor traffic (*i.e.* P2P, FTP, IM, and Web). HMMs are employed

to build inbound and output models of the application types considered, and are fed with features based on burst volumes and directions of Tor flows, obtaining an accuracy up to 92%. AlSabah *et al.* [149] present a ML-based method employing NB, Bayesian Network (BN), functional and logistic model trees to recognize applications used by Tor users. Both *circuit-level* and *cell-level* information is leveraged for offline and offline/online classification, respectively. The highest accuracy achieved in online (resp. offline) case equals 97.8% (resp. 91%). A similar setup is proposed in [150] for user activity recognition by means of four classifiers (*i.e.* NB, BN, RF, and C4.5) fed with *traffic-flow* and *circuit-level* features. Both approaches reach $\approx 100\%$ accuracy, with flow-based TC being *less demanding* and based on data that could be captured anywhere between the user and the Tor's relay. Along the same lines, Shahbar and Zincir-Heywood [151] employ flow-based (statistical) traffic analysis to prove whether Tor PTs can evade censorship systems. Adopting a C4.5 classifier and based on a thorough analysis, the authors show that Tor PTs' usage is recognizable, as PT-based obfuscation changes the content shape in a distinct way with respect to "normal" Tor (*i.e.* conferring to flows distinctive fingerprints). The effects of bandwidth sharing on I2P is analyzed by the same authors in [152] considering both application and user profiling achievable by an attacker. Using a C4.5 classifier fed with flow-based features, the results show that users and applications on I2P can be profiled, with a harmful (resp. beneficial) effect of the shared bandwidth increase on applications (resp. users) profiling accuracy.

Recently, Shahbar and Zincir-Heywood [44] describe Anon17 public dataset comprising directional traffic-flows obtained by collecting data from three ATs (*i.e.* Tor, I2P, and JonDonym). Besides, detailed information about the traffic types and applications running on Tor and I2P, such as "Browsing" and "EEpsites", respectively, as well as the PTs employed on

the Tor network, is provided in the form of *three-level labels* for each flow. Up to our knowledge, the sole public dataset similar to Anon17 is that described in [110], containing however only Tor traffic of eight applications (*i.e.* browsing, audio, chat, mail, P2P, FT, VoIP, and video) and providing only time-related features.

4.2 Hierarchical Traffic Classification Approach

In this section, we introduce the proposed framework for HC of encrypted and anonymous traffic, as streamlined in Fig. 4.1, whose main components are discussed in the following. In detail, preliminaries on HC are discussed first, together with a general overview of the proposed approach (§4.2.1). Then, we introduce the possible traffic object choices, along with a description of the feature sets (§4.2.2). The section ends with the classification algorithms (§4.2.3) which could be adopted for hierarchical analysis of ATs' traffic.

4.2.1 Preliminaries and Proposed HC Framework

In what follows, we provide HC preliminaries needed to understand the design choices adopted for the general HC framework here proposed and investigated in the context of TC of ATs.

Firstly, we remark that our classification framework focuses on classes whose relationship can be summarized in the form of a tree (*i.e.* each class has one parent class, *at most*) with T classification *levels* and L_t classes to discriminate from at the t^{th} level (whose number increases with the depth), organized in the corresponding set $\mathcal{L}_t \triangleq \{1, \ldots, L_t\}$.

The tree structure can be explored with three alternative approaches [153]: (i) top-down approach, when the system employs a set of local classifiers

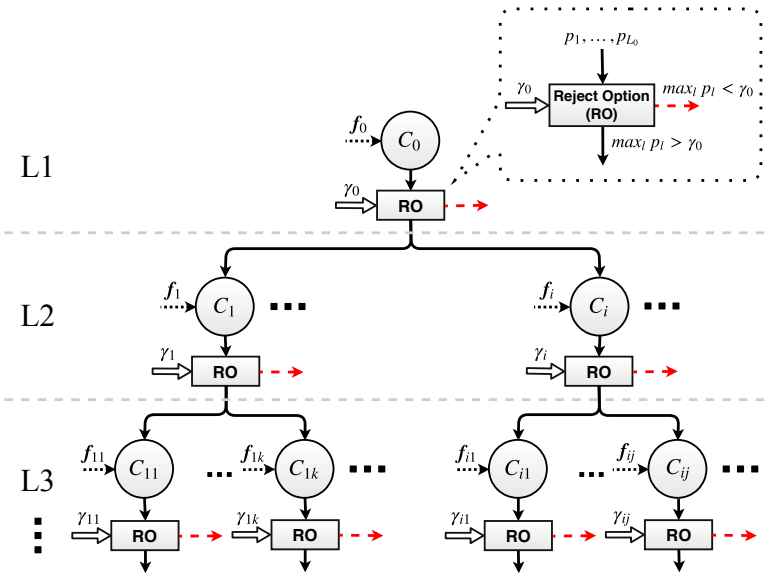


Figure 4.1: Sketch of the proposed HC framework.

each tackled to a specific sub-problem; (*ii*) big-bang (or global) approach, when a single classifier copes with the entire class hierarchy (*i.e.* it is trained by considering the entire class hierarchy at once); (*iii*) flat classification approach, ignoring the class relationships at different levels (*i.e.* solving a classification task at each level of the hierarchy in an independent fashion). We have chosen to adopt the *top-down* choice (similarly to [77]). In this case, for each instance to be classified, the HC framework firstly predicts its first-level (most generic) class, then it uses that predicted class to narrow the choices of classes to be predicted at the second level (*i.e.* allowed second-level predicted classes are the children of that predicted at the first level), and so on. Although errors at a certain class level could propagate downwards the hierarchy, this choice promotes the *architecture modularity*, which is crucial in TC, as opposed to big-bang and flat classifiers.

Further, *three different ways of using the local information can be employed*, mainly differing in their training phase [153]: (*i*) Local Classifier per Node (LCN); (*ii*) Local Classifier per Level (LCL); (*iii*) Local Classifier per Parent Node (LCPN). LCN and LCL consist in training one binary classifier for each node and a multi-class classifier for each level of the class hierarchy, respectively. Unluckily, since these two approaches suffer from class-membership inconsistency and have higher complexity, we have chosen to adopt *LCPN*, that is widely used in the literature [77, 88] and requires a multi-class classifier for each parent node in the class hierarchy, trained to distinguish among its children nodes that are usually less than L_t, with t being the node classification level.

Figure 4.1 reports a sketch of the proposed HC framework. We highlight that the indexing of a node reflects the ordered list of its ancestors (except the *root* C_0); for example C_{ij} denotes the j^{th} L_3 classifier having C_0 and C_i, as grandparent and parent, respectively. The classifier C_{ij} is in charge of discriminating from $\bar{L}_{ij} < L_3$ classes, grouped within the set Ω_{ij}, based on the associated prediction probabilities $p_1, \ldots, p_{\bar{L}_{ij}}$.

The proposed HC framework is trained by *recursively* splitting the training set according to the tree structure. Specifically, the procedure starts from the root classifier C_0 trained using the whole set. On the other hand, each node concurring to $t > 1$ level classification uses a training set corresponding to a subset of $\mathcal{L}_t$, the elements all belonging to the same class at $(t-1)$. Reducing the number of classes per classifier hopefully simplifies the resulting problem and reduces the error scope of flat classification.

In addition, the proposed framework adopts a *progressive-censoring* (viz. non- mandatory leaf node prediction [153]) policy, accomplished by equipping each classifier node with a "reject option" that censors "unsure" classification outcomes at intermediate layers ("RO" blocks in Fig. 4.1). As

introduced in Sec. 1.3.2, the reject option forces the classification process to stop for a given instance when a classifier node (*e.g.*, $C_{i,j}$) does not reach a clear verdict, *i.e.* when the highest class prediction probability (*e.g.*, $\max_{\ell=1,\dots,\bar{L}_{i,j}} p_\ell$) is below a threshold (*e.g.*, γ_{ij}). This design choice here avoids that misclassifications are propagated downwards, at the expense of coarser-grained predictions.

By looking at the hierarchy of classifiers reported in Fig. 4.1, it is apparent that its naïvest implementation resorts to the *same* classification algorithm and feature set throughout all the hierarchy. Nonetheless, although HC can achieve a potential performance gain with respect to a flat approach even in this case, the proposed framework allows for further *optimization* [153], due to the decomposition of the classification task into sub-problems. Indeed, classification performance is expected to improve with more refined implementations, leveraging a *specific selection of features for classification*, and/or using *different classification algorithms at different nodes of the class hierarchy*, chosen, for example, from a pool of available classifiers. Therefore, we investigate both these optimization degrees-of-freedom. In the case both the classifier and the feature set are optimized at each node, the combinatorial explosion of the resulting optimization is herein circumvented by a decoupled design resorting to *per-node performance*, that is selecting the pair corresponding to the classifier and the number of features ensuring the highest score for the sub-classification problem the classifier node is in charge to solve. Nonetheless, we remark that an optimization based on complete enumeration or alternative heuristics does not contrast with the HC architecture depicted in Fig. 4.1.

4.2.2 Traffic Object and Feature Design

Although several proposals exist and have been considered in the (anonymous) TC literature, *flows* and *biflows* are the most commonly used traffic objects. Referring to the hierarchical approach illustrated in Fig. 4.1, we assume that all the classifiers in the hierarchy shall *operate on a common TC object*, although not restricted to a specific one.

As previously explained, different sets of features (of different sizes) can be considered to feed the classifiers in the hierarchical architecture, as shown in Fig. 4.1, with the aim to achieve accurate TC. For instance, referring to the example in Sec. 4.2.1, the classifier C_{ij} is assumed to rely on M_{ij} features, collected in the vector $\boldsymbol{f}_{ij}$. Finally, given a set of features, *feature selection/extraction techniques* are adopted to extract only a subset among them, with the aim of improving further TC performance, while reducing the computational complexity.

4.2.3 Classification Algorithms

As in the case of the set of features, the proposed HC approach allows for a different classifier to be employed at each node (see Fig. 4.1). Therefore, any ML/DL-based (that can deal with the encrypted nature of ATs' traffic) supervised classifier can be adopted. For example, referring to the classifier C_{ij} in the example of Sec. 4.2.1, any ML/DL-based classifiers could be used to discriminate from $\bar{L}_{ij}$ classes within the set Ω_{ij}. The sole requirement for each classifier is to be able to provide its soft-output vector, required by the censoring mechanism described in Sec. 4.2.1.

4.3 Experimental Environment

Hereinafter, we first describe the scenario in which we have carried out our analyses (§4.3.1); then we discuss the design choices made to implement the HC system, matching the characteristics of the Anon17 dataset (§4.3.2); finally, we introduce the performance metrics employed to evaluate our solution (§4.3.3).

4.3.1 HC Scenario

To design, implement, and evaluate our HC approach, we have exploited the publicly-available Anon17 dataset, whose details are reported in Sec. 2.5.2. According to the Anon17 description, we tackle the (hierarchical) classification of anonymity networks, traffic types, and applications under the assumption that we are in presence of anonymous traffic only, based on a two-fold motivation. First, the depicted scenario refers to an application context in which an *upstream classifier* has been able to provide an accurate screening of clear and standard-encrypted traffic and then separate this latter from ATs' traffic, as demonstrated, for example by Barker *et al.* [146] and more recently by Rao *et al.* [154] for the Tor network. Hence, the aim of the proposed approach is to assess discrimination of anonymity services and related applications *once this ATs' traffic has been separated from other traffic*. Indeed, the use of an upstream classifier would perfectly fit within the hierarchical approach proposed, representing a further step toward the development of a whole HC framework for traffic analysis of clear/encrypted/anonymous data, truly operating in an open-world scenario. Secondly, the results of our analysis can be intended as an upper bound on the ATs' classification performance in the case of an *open-world* assumption. Indeed, a negative answer to our question (*i.e.* an unsatisfactory performance

in classifying anonymous traffic only) would lead to the conclusion that anonymous traffic, even though perfectly screened from the remaining traffic bulk, would still remain an unobservable black-box to an eavesdropping user. Our results will show that this is not the case, and confirm that there is room for classification of ATs in an open-world assumption.

4.3.2 HC Implementation Choices

Hereinafter, we briefly discuss how the general HC framework described in Sec. 4.2 is specialized in the case of its adoption on the Anon17 dataset, with the following paragraphs covering all the different aspects needing specification. The HC framework specialized for the classification of Anon17 traffic-flows has been implemented in *Python*, leveraging *Weka* wrapper and *Scikit-learn* utilities. A Virtual Machine (OS Ubuntu Server 16.04) equipped with 32 VCPUs and 64 GB RAM, running on private *OpenStack* cloud platform, has been employed for evaluating the HC framework. Finally, the code used for the analyses has been made available on GitHub[1] to foster reproducibility of the experiments.

HC Architecture. Based on the nature of the traffic in the Anon17 dataset, the TC here considered is arranged in three levels, corresponding to anonymity networks, traffic types, and specific applications. As a whole, the HC framework resorts on *one* classifier at L_1, *two* classifiers at L_2, and *five* classifiers at L_3.

Traffic Object and Feature Sets. Anonymous traffic contained in Anon17 is split into different *flows*, by means of the flow-exporting tool *Tranalyzer2* [155], constituting the traffic object here employed. We highlight

[1] https://github.com/NM2/hierarchical-tc-at.

that the direction of each flow (considered as a feature) is indicated as "A" or "B" for client-to-server and vice versa, respectively. We note that the proposed HC approach could even operate at the *packet-level*, in principle[2] (viz. the traffic object could be the *single packet*).

For each traffic flow, Anon17 provides different sets of features extracted via Tranalyzer2 [155]. More specifically, the latter is an open source tool that generates flows from a captured traffic dump or directly by working on the network interface, based on the *libpcap* library. Tranalyzer2 is bundled with different basic plugins, being able to extract a plethora of features per flow. However, the dataset provides only a subset of these features, since some of them have been removed (*e.g.*, ICMP and VLAN features) because they do not provide useful fingerprinting information. Additionally, aiming at protecting users' privacy and simulate a true encrypted-traffic scenario, IP addresses and payloads of the packets have also been removed from the dataset. In our HC framework, we consider *two* different feature sets, referred hereinafter to as `TC_set` and `EarlyTC_set`. In brief, `TC_set` capitalizes complete traffic flows, while `EarlyTC_set` only relies on the first K packets of each flow, thus enabling early-TC.

Specifically, `TC_set` originally refers to 81 per-flow statistical features. Firstly, we have performed dataset sanitization to remove the fields `min_pl`, `max_pl`, and `mean_pl`, as they seem repeated with respect to `minPktSz`, `maxPktSz`, and `avePktSize`, respectively, considering the specific configuration adopted in Tranalyzer2 for capturing the traffic. Additionally, we have discarded the initial/final timestamps of each flow (`time_first` and `time_last`) to avoid biased results, as this pair of features may be influenced by a sequential collection of the traffic traces belonging to different

[2]Up to our knowledge, there is no work in literature tackling anonymous TC at this granularity.

ATs and/or application types, potentially introducing classification artifacts. Therefore, we have considered a reduced set of 76 fields, that has been exploited to extract *three different types of features belonging to the* `TC_set`. The *first* type of features considered comprises 74 summarizing flow-based statistics, such as:

- Flow direction and duration.
- Packet Length (PL) statistics (mean, min, max, median, quartiles, etc.).
- Inter-Arrival Time (IAT) statistics (mean, min, max, median, quartiles, etc.).
- TCP header-related features[3] (window size, sequence number, TCP options, etc.).
- IP header-related features (type-of-service, time-to-live, IP flags, etc.).
- Number of Tx/Rx bytes and packets (including bytes/packets Tx rate and stream asymmetry measures).
- Number of distinct hosts connected to flow source or destination IP during its lifetime.
- Number of concurrent flows sharing the same (source IP, destination IP) pair regardless of source & destination ports.

We point to the user manual of Tranalyzer2 [156] for further details about these features. The *second* and *third* type of `TC_set` features are based on a finer histogram representation of PL and joint PL-IAT, respectively.

[3]TCP-related features have zero-value if the flow leverages UDP as transport protocol (*e.g.*, I2P network works on both TCP and UDP).

These sets are obtained through an appropriate format conversion and/or marginalization [157] from the Anon17 field `ps_iat_histo`. This field, as provided by Tranalyzer2, contains precise (non-binned) PL info, whereas applies a non-uniform binning to the (continuous-valued) IAT information. More precisely, the IAT range is divided in 91 bins with the following ranges: bins $0-39$ covering $[0,200)$ ms with 5 ms width, bins $40-59$ covering $[200,400)$ ms with 10 ms width, bins $60-89$ covering $[400,1000)$ ms with 20 ms width and bin 90 for IAT values higher than 1 s [156]. Their use will be investigated similarly to [158] to understand whether finer-grained features can improve classification performance.

Differently, `EarlyTC_set` is made of the sequence of pairs `(PL, IAT)` of the first K packets of each flow. This set is extracted from the field `nfp_pl_iat`[4], containing the info corresponding to the first $K = 20$ packets, as set by Tranalyzer2 default options [156]. In the rest of the paper, we will employ `TC_set` when referring to standard TC, whereas the adoption of `EarlyTC_set` will be assumed just for early TC.

Finally, for the first type of features belonging to `TC_set`, we consider feature selection, based on a *filtering approach*, ranking the elements of the set based on the relative importance of each feature, so as to skim the more informative ones, in terms of *mutual information* with the class (random) variable. The aim is to possibly improve further their performance, while reducing their computational complexity. We do not consider *wrapper methods* since they may be considerably more complex and coupled to a specific classification algorithm [159]. We also remark that other feature selection measures have been also tried (*e.g.*, the Pearson's correlation or the symmetric uncertainty), obtaining similar trends and slightly worse perfor-

[4]Only in this case, the `pl` acronym is referred to the Payload Length (and not to the Packet Length), in accordance to Tranalyzer2 nomenclature [156].

mance. On the other hand, for `EarlyTC_set` features, ranking is performed according to a time-constraint (*i.e.* only the first K packets are employed). We remark that, for the `TC_set`, feature extraction techniques, such as Principal Component Analysis (PCA), could be easily adopted in the proposed HC framework without any substantial changes.

Classification Algorithms. Herein, we consider as potential nodes *five* different ML-based classifiers, namely the (*i*) C4.5, the (*ii*) RF, the (*iii*) NB, the (*iv*) MNB and the (*v*) BNs. Indeed, these classifiers have been successfully employed in several works tackling TC of anonymous traffic [149, 150, 152, 160]. Nonetheless, as the proposed HC framework is general, other ML (*e.g.*, SVM, Gradient Boosting, etc.) or even DL classifiers (cf. Chapter 5) could be adopted with no substantial change. Specifically, the *first two* belong to the family of *decision trees*, whereas the *latter three* to the *Bayesian family*. We underline that the base classifiers employed in the complementary MC framework described in Chapter 3 belong to the same classifier families (they may differ in the feature set / configuration adopted being dependent from the specific context / dataset). In the following, we recall the base principles of the classification algorithms leveraged.

1. *C4.5*: it is an algorithm employed to generate a decision tree used (mainly) for classification purposes [161], based on the concept of entropy of a distribution [157]. The training algorithm obviates to the NP-hardness of optimal tree search by means of a greedy procedure, based on a top-down recursive construction. Then, instances are partitioned recursively based on the chosen feature whose values most effectively split so as to maximize a purity measure in the data, such

as the "gain ratio", that avoids bias toward features with a larger support [161]. Thus, the splitting criterion is triggered by the feature ensuring the highest gain ratio (*i.e.* purity). C4.5 recurs on the smaller sub-lists, until the following termination criteria are met: (a) all the instances in the list belong to the same class (a leaf node is here created with a label associated to that class); (b) there are no remaining features for further partitioning (each leaf is labeled with the majority class in the subset); (c) there are no examples left. C4.5 introduces also refinements introduced to reduce over-fitting, including pre-pruning (*i.e.* stop growing a branch when information becomes unreliable) and post-pruning (*i.e.* growing a decision tree that correctly classifies all training data and then simplify it later by replacing some nodes with leaves), with the latter preferred in practice.

2. *Random Forest (RF)*: it is a classification method based on an ensemble of B several decision trees (it is a free parameter tuned by cross-validation or via the "out-of-bag" error), built at training time exploiting the ideas of "bootstrap aggregating" (bagging) and random-feature selection to mitigate over-fitting [134]. Specifically, during the training phase, each decision tree in the RF classifier is grown based on a bootstrap (*i.e.* a uniformly random sampling procedure with replacement) sample set of the training data available. To further reduce overfitting the RF adds to the above scheme a modified tree learning algorithm named "feature bagging" that selects, at each candidate split in the learning process, only a random subset (whose size is another free tunable parameter) of the features. Finally, after training, decision on testing samples can be made by taking the majority vote or soft combination of the responses of B trees.

3. *Naïve Bayes (NB)*: it is a simple probabilistic classifier that assumes class conditional independence of the features, being not the case for real-world problems, but working well in practice and leading to reduced complexity. More specifically, the NB evaluates the probability that an unlabeled test instance $\boldsymbol{f}_T$ belongs to each class c_i, namely the posterior probability $P(c_i|\boldsymbol{f}_T)$ through the Bayes' theorem, and returns the label corresponding to the maximum posterior among the classes, that is $P(c_i|\boldsymbol{f}_T) \propto P(c_i) \prod_{m=1}^{M} P(f_{T,m}|c_i)$, where "$\propto$" means proportionality and $P(c_i)$ denotes the a-priori probability of class c_i (estimated from the training set). On the other hand, each distribution $P(f_m|c_i)$ is estimated by resorting to a PMF when the feature is categorical, whereas common alternatives for numerical features include: (a) Moment Matching to a Gaussian PDF (NB), (b) Supervised Discretization (NB_SD), and (c) Kernel-based Density Estimation (NB_KDE) [126].

4. *Multinomial Naïve Bayes (MNB)*: this classifier adopts sample histograms as a different set of features and treats these features as frequencies of a certain value of a categorical random variable, comparing the sample histogram of each test instance with the aggregated histogram of all training instances per class. Therefore, the evaluation of the conditional PMF $P(\boldsymbol{f}_T|c_i)$ is proportional to $\prod_{m=1}^{M}(\rho_m)^{f_{T,m}}$, where ρ_m indicates the probability of sampling the m^{th} feature. On the basis of the results of Chapter 3, we will employ a variant of the MNB classifier, adopting *term frequency transformation* with *cosine normalization*, exploited also in [65] for website fingerprinting in anonymous networks.[5]

[5]It is worth noting that preliminary investigations have also confirmed the highest per-

5. *Bayesian Networks (BNs)*: they are graphical representations which model dependence relationships between features and classes [162], collectively represented as the set of random variables $\boldsymbol{U} \triangleq \{f_1, \ldots, f_M, C\} = \begin{bmatrix} U_1 & \cdots & U_{M+1} \end{bmatrix}^T$. Unlike the NB classifier, they are not based on the conditional independence assumption for the features. Formally, a BN for $\boldsymbol{U}$ is a pair $\mathcal{B} \triangleq \langle \mathcal{G}, \Theta \rangle$, which is learned during the training phase. The first component ($\mathcal{G}$) is a *Directed Acyclic Graph* that encodes a joint probability distribution over $\boldsymbol{U}$, where each vertex represents a random variable among $U_1, \ldots, U_{M+1}$ and the edges represent their dependencies. The second component (Θ) represents the set of parameters modeling the BN, uniquely determining the local conditional distributions associated to the BN, which allow to encode the joint distribution $P_{\mathcal{B}}(f_1, \ldots, f_M, C)$. Finally, during the testing phase, for each instance $\boldsymbol{f}_T$, the BN returns the label $\hat{c} \triangleq \arg\max_{c_i \in \Omega} P_{\mathcal{B}}(c_i | \boldsymbol{f}_T)$, based on the Bayes' theorem. We will either consider a BN classifier with (default) *K2 search* (BN_K2) for structure learning, or impose the network to have a *tree-augmented form* (BN_TAN) where the tree is formed by calculating the maximum weight spanning tree.

In the following analyses, all the classifiers will be fed either with the *first* type of features belonging to the `TC_set` (*i.e.* flow-based statistics) or with the features of the `EarlyTC_set` (*i.e.* sequence of (Payload Length, IAT) of the first K packets), with the sole exception of the MNB that, working on features in the form of histograms, will be fed with the *second* (*i.e.* PL histogram) and *third* (*i.e.* joint PL-IAT histogram) types of `TC_set` features.

formance of the considered variant with respect to the others [65] analyzed in Chapter 3.

4.3.3 Performance Metrics

In addition to common performance metrics described in Sec. 1.3.2, namely accuracy, F-measure, and G-mean, along with confusion matrices, to detect *performance bottlenecks* of our HC approach, we also provide *per-node metrics* (*i.e.* not considering classification errors introduced by upper levels), also deriving useful guidelines for system design and evaluation. Considering that in our design each classifier implements a *reject option*, we also deepen the impact of this design choice on performance. In more detail, we investigate the impact of varying the censoring thresholds (see §4.2.1), whose tuning can be effective to improve classification performance trading it off with the reduction of the classified instances (viz. the ratio of classified instances, CR).

Finally, for each considered analysis, our evaluation is based on a *(stratified) ten-fold cross-validation*, representing a stable performance evaluation setup. As a consequence, we report both the mean and the standard deviation (in the form of a $\pm 3\sigma$ interval, corresponding to 99.7% confidence under a Gaussian assumption) of each performance measure as a result of the evaluation on the ten different folds.

4.4 Experimental Evaluation

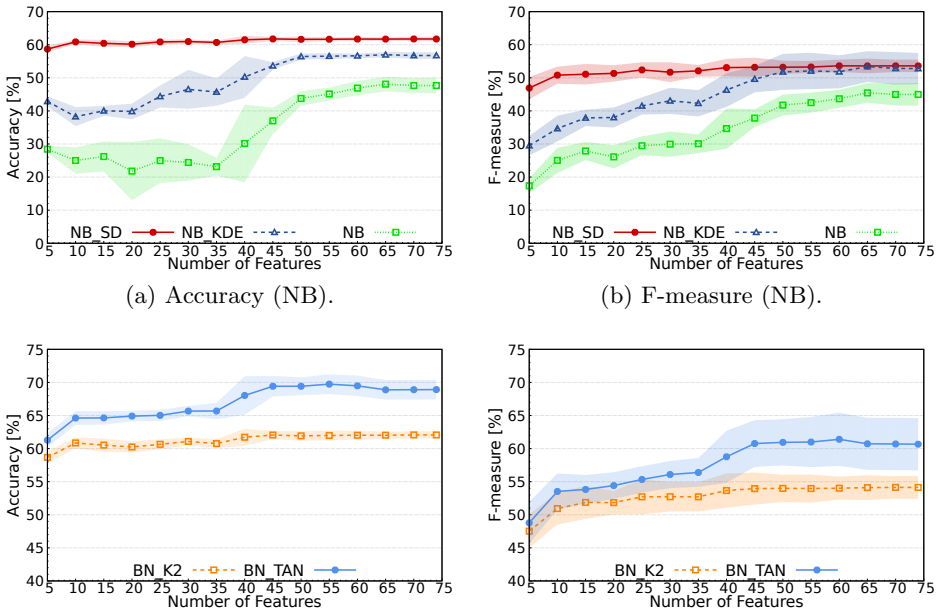
In this section, we show experimental results aimed at investigating anonymous TC performance via the proposed hierarchical framework, also when considering early TC. First, we identify the best-performing flat classifier fed with `TC_set` features (§4.4.1), considering both different classifiers' variants (*e.g.*, for the NB and BNs) and various types of feature (*e.g.*, histogram-based features for the MNB). Then, we analyze the performance of the proposed hierarchical approach and compare it with the best flat classifier,

performing an optimization of the hierarchical framework in different steps (§4.4.2) both in the flow-based (*i.e.* `TC_set` features) and in the early-TC scenario (*i.e.* `EarlyTC_set` features). Regarding this latter scenario, since the difference with the `TC_set` analysis mainly concerns the feature selection performed, only the final results pertaining to fine-grained optimization (along with the comparison with the best flat alternative) are reported. Finally, we perform a detailed evaluation that contains finer-grained analyses of the error patterns of these two different classification "philosophies" along with the severity of errors that leads to interesting conclusions on the ATs considered (§4.4.3). Also, this analysis encompasses a first investigation of classification performance obtained by resorting to progressive censoring in the hierarchical case that we compare with the effects of censoring on a flat classifier baseline.

4.4.1 Selecting the Best Flat Classifier

In the following, we preliminarily investigate flow-based TC leveraging the features belonging to the `TC_set` (*i.e.* comprising summarizing flow-based statistics and PL/PL-IAT histograms) with the aim of selecting the best flat classifier to use as baseline for the successive analyses. However, before proceeding with a rigorous comparison of the classifiers here considered, we firstly focus on relative performance evaluation of the different variants of NB and BN taken into account in Sec. 4.3.2.

To this end, Fig. 4.2 shows the accuracy and F-measure of NB, NB_SD, and NB_KDE (resp. BN_K2 and BN_TAN) in top (resp. bottom) plots. Note that the results pertain to L_3, being the *hardest* classification task, but similar trends have been observed also for the other two levels. Furthermore, the performance has been evaluated by training/testing the classifiers with a varying subset of features, ranked in decreasing importance as described



(a) Accuracy (NB). (b) F-measure (NB).

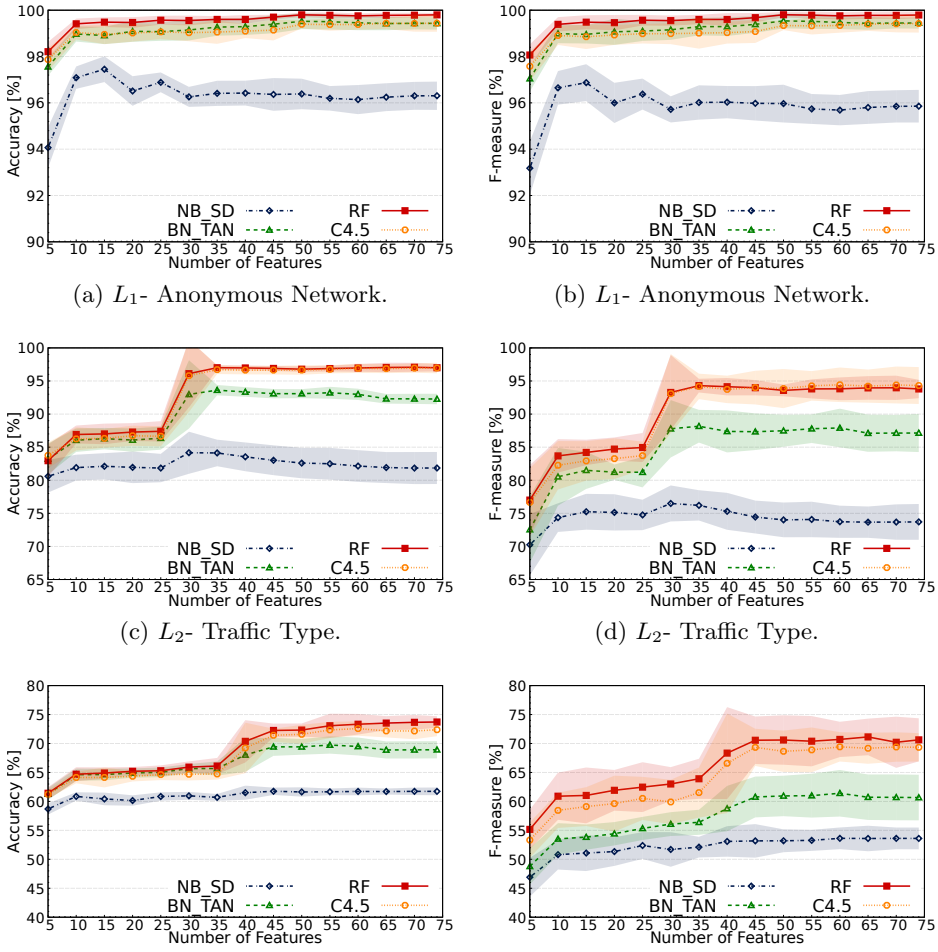
(c) Accuracy (BN). (d) F-measure (BN).

Figure 4.2: Accuracy and F-measure of NB (a-b) and BN (c-d) classifiers for different subsets of features (from 5 to 74 with increments of 5) for L_3 (Application) level. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

in Sec. 4.3.2 so as to draw general conclusions.[6]

From the inspection of the figure, it is apparent that both NB_SD and NB_KDE outperform NB over all the range of feature subsets, with NB_SD achieving higher performance even in the case of a smaller set. Similarly, BN_TAN significantly outperforms BN_K2. The former result can be explained as density estimation of each feature, either in a discretized (NB_SD)

[6]Using the Scikit-learn estimator `mutual_info_classif`, employed in conjunction with the `SelectKBest` ranker which allows obtaining the top $K_\star$ (most informative) features, with $K_\star$ as input parameter.



(a) L_1- Anonymous Network.

(b) L_1- Anonymous Network.

(c) L_2- Traffic Type.

(d) L_2- Traffic Type.

(e) L_3- Application.

(f) L_3- Application.

Figure 4.3: Accuracy (a, c, e) and F-measure (b, d, f) of flow-based classifiers for different subsets of statistical `TC_set` features (from 5 to 74 with increments of 5) for each classification level. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

or "kernelized" (NB_KDE) fashion, is beneficial since the Gaussian hypothesis represents an overly simplified assumption, whereas in the latter case it is apparent that constraining BN structure learning to a tree-augmented form (as opposed to a greedy sub-optimal learning) provides improved generalization capabilities. Thus, in the remainder of this section, only the variants NB_SD and BN_TAN will be considered in our comparison, being the best-performing Naïve Bayes and Bayesian Networks classifier variants observed, respectively.

Then, with the aim of selecting the best subset (viz. an optimized number) of features and provide a comparison of the supervised techniques considered, in Fig. 4.3 we show the performance of all the classifiers considered—except for the MNB, whose performance relies on histogram-based features and will be thus discussed later—when varying the ranked subset of features for both training and test sets. As apparent from the results, all the classifiers obtain excellent results in L_1 classification, that is all achieve both $> 95\%$ accuracy and F-measure when approximately only the top 15 features are employed. On the other hand, performance metrics generally degrade with the increasing granularity of the classification task (*i.e.* moving from L_1 to L_3). This intuitive trend can be attributed to the increasing difficulty of the classification task being tackled. Indeed, the discrimination of anonymous traffic at L_3 is harder than trying to discern merely the anonymity network. Interestingly, the degradation level varies with the classifier and it is observed to be milder for C4.5 and RF, whereas it is the highest for NB_SD. This finding can be explained as the conditional independence assumption of the features for NB_SD is limiting when tackling harder classification tasks (*i.e.* L_3). Overall, from the inspection of the figures, we can notice that the performance of all classifiers (approximately) reaches a steady value when using a number of features that also depends

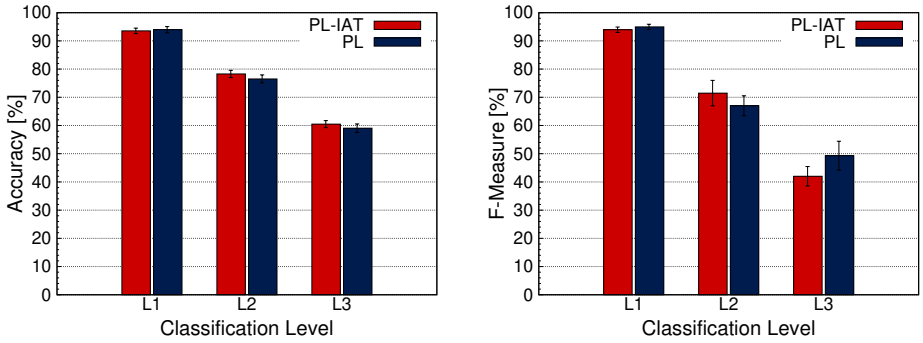
Table 4.2: Best overall accuracy and macro F-measure for dataset $\bar{D}_5$ obtained with different [optimal number of features] employed. Highlighted values: **maximum accuracy** and maximum F-measure for each level.

Classifier	**Metric**	L_1	L_2	L_3
NB_SD	*Accuracy*	97.45% [15]	84.16% [30]	61.75% [45]
	F-measure	96.87% [15]	76.49% [30]	53.62% [60]
BN_TAN	*Accuracy*	99.52% [50]	93.61% [35]	69.74% [55]
	F-measure	99.54% [50]	88.16% [35]	61.40% [60]
C4.5	*Accuracy*	99.44% [74]	96.96% [74]	72.58% [60]
	F-measure	99.42% [74]	94.43% [60]	69.42% [60]
RF	*Accuracy*	**99.80%** [50]	**97.01%** [35]	**73.52%** [65]
	F-measure	99.80% [50]	94.30% [35]	71.14% [65]

on the classification granularity, with harder TC tasks requiring more features to reach better performance. Indeed, around the top 15, 30, and 45 features are needed at L_1, L_2, and L_3, respectively.

Collectively, with an optimized number of features, the highest performance is obtained by RF and (in a single case) C4.5, corresponding to 99.80% (resp. 99.80%), 97.01% (resp. 94.30%) and 73.52% (resp. 71.14%) at L_1, L_2, and L_3, respectively, in terms of accuracy (resp. F-measure), as shown by the summarizing results reported in Tab. 4.2.

We now focus on investigating whether (*i*) finer-grained features, such as histograms, would improve performance and (*ii*) whether these finer-grained features would require time-related information. We recall that the appeal of histogram-based features has been highlighted by different works on TC [158, 163]. Based on this reason, in Fig. 4.4 we report the performance (in terms of both accuracy and F-measure) of MNB in conjunction with the use of the *second* (PL histogram) and *third* (joint Payload Length & Inter-Arrival Time histogram) type of `TC_set` features. Similarly to the previous



(a) Accuracy. (b) F-measure.

Figure 4.4: Accuracy (a) and F-measure (b) of MNB classifier [65] leveraging PL and PL-IAT histograms as features. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

analyses, the performance is evaluated at the three levels of granularity for the sake of a complete comparison. First, it is apparent that considering histogram-based features does not improve classification performance, as evident from comparison of Fig. 4.3 and Fig. 4.4. The lack of improved performance may be due to a two-fold reason: (*i*) when using the MNB, the features pertaining to IP/TCP headers and number of connections are not taken into consideration and (*ii*) the histogram discretization provided by Anon17 may be not adequate for developing an accurate fingerprint. Interestingly, time-related features do not improve appreciably classification performance at the first two levels. The only exception is represented by the increase of F-measure at L_3 (+6.85%) when the PL histogram feature set is employed, highlighting a detrimental effect of time-related features. This trend may be attributed to the harder classification task to be solved.[7]

[7]Given this result, in the following, we will refer with `TC_set` features to only the *first* type (*i.e.* per-flow statistics), if not stated otherwise.

Based on the outcome of these analyses, we select and employ the best-performing flat classifier (*i.e.* the RF with different feature configurations) as both baseline and starting point for the optimization of the proposed hierarchical classification architecture as described in the following section. We will also give further considerations on the early-TC scenario in a dedicated paragraph.

4.4.2 Optimizing Hierarchical Classification Framework

The next paragraphs show increasingly advanced optimizations of the proposed hierarchical TC framework. First, we demonstrate that the usage of a "naïve" hierarchical architecture is already able to introduce non-negligible improvements. Successively, the results of design enhancements are shown, deepening the impact of both rough- and fine-grained optimization choices—the former consisting in varying the number of features of the classifiers (keeping the classifier type fixed) in the hierarchy with the same increment, while the latter involving changes to both features and classifier types.

Naive Hierarchical Framework. In Tab. 4.3 we report the performance of the best flat classifiers (*e.g.*, the best configurations with an optimal number of features in terms of accuracy) as derived from Sec. 4.4.1 and resulting in the RF fed with 50, 35, and 65 features at L_1, L_2, and L_3, respectively (see Tab. 4.2). In the following, we show the performance in terms of accuracy, F-measure, and G-mean at each classification level.[8] Such optimal setup is compared with a first naïve implementation of our HC approach, obtained by using the best L_3 flat configuration (*i.e.* the RF fed with the best 65 features) in *all* the classifier nodes of the hierarchy, namely

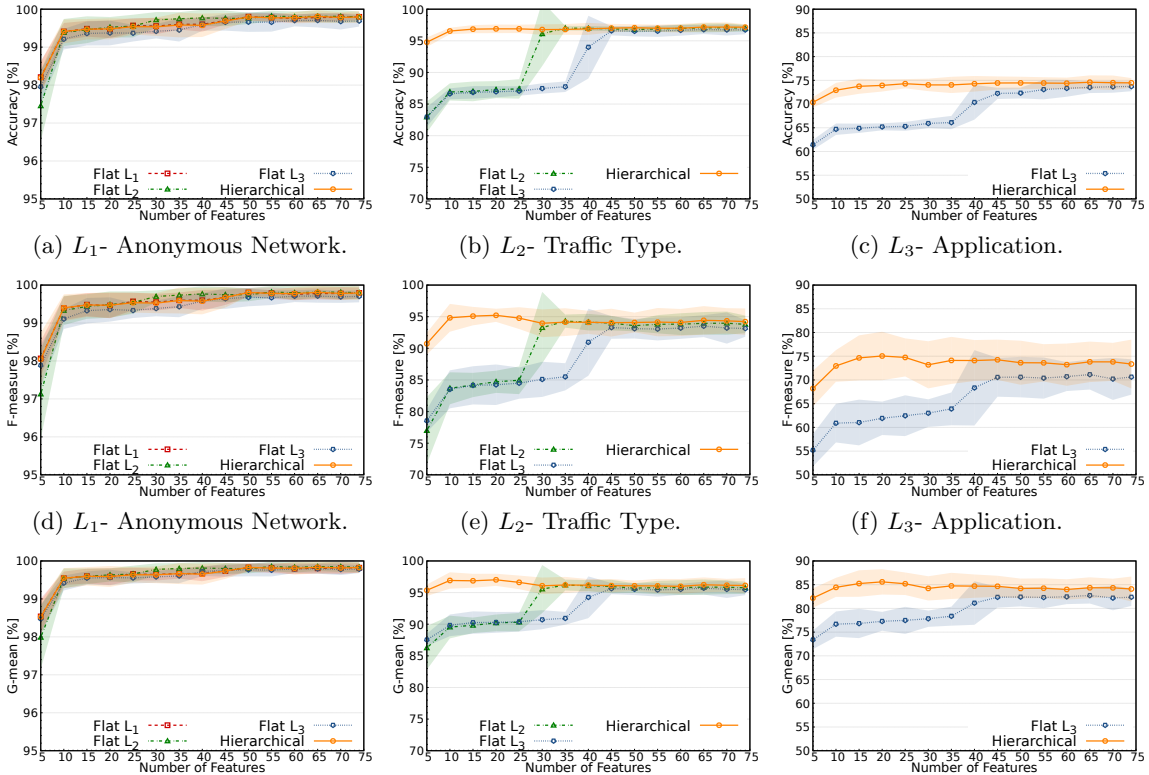
[8] "n.d." points out unavailable performance for flat classifiers at levels deeper than that considered for classification.

Table 4.3: Accuracy, F-measure, and G-mean (%) of the best flat classifier (RF) with [optimal number of features] at each level compared to naive hierarchical configuration. Results are in the format *avg. (± std.)* over 10-folds. Highlighted values: Best Accuracy (⋆), F-measure (†), and G-mean (♢) per level.

Classifier	Metric	L1	L2	L3
Best Flat L1 [50]	*Accuracy*	99.80±0.03%	n.d.	n.d.
	F-measure	99.80±0.04%	n.d.	n.d.
	G-mean	99.83±0.03% ♢	n.d.	n.d.
Best Flat L2 [35]	*Accuracy*	99.75±0.06%	97.01±0.24%	n.d.
	F-measure	99.73±0.06%	94.30±0.35%	n.d.
	G-mean	99.80±0.05%	96.19±0.29%	n.d.
Best Flat L3 [65]	*Accuracy*	99.70±0.06%	96.77±0.24%	73.52±0.40%
	F-measure	99.71±0.06%	93.51±0.58%	71.14±1.05%
	G-mean	99.79±0.04%	95.71±0.34%	82.73±0.57%
Naive Hierarchical [65]	*Accuracy*	99.81±0.06% ⋆	97.17±0.24% ⋆	74.60±0.48% ⋆
	F-measure	99.81±0.06% †	94.43±0.75% †	73.82±1.42% †
	G-mean	99.83±0.05%	96.23±0.39% ♢	84.35±0.74% ♢

it is the result of the decomposition in *simpler TC sub-tasks*.

Unsurprisingly, L_1 performance metrics report a score $\geq$ 99.7%: the traffic generated through different anonymity networks is easily distinguishable from each other, confirming that these tools are designed to provide anonymity but not to hide the usage of the tool itself. Moreover, the results show that even a naïve hierarchical solution improves L_3 performance (up to +2.68% in terms of F-measure) with respect to the best L_3 flat classifier, and performs on a par in terms of L_1–L_2 levels when compared to the level-optimized best flat classifiers. Also, from a statistical significance viewpoint, we observed a gain of the HC approach in 100% of the cases over the considered folds. This overall improvement is due to the split of the original TC task (21 classes at L_3, see Fig. 2.8) into smaller tasks, at most



(a) L_1- Anonymous Network. (b) L_2- Traffic Type. (c) L_3- Application.

(d) L_1- Anonymous Network. (e) L_2- Traffic Type. (f) L_3- Application.

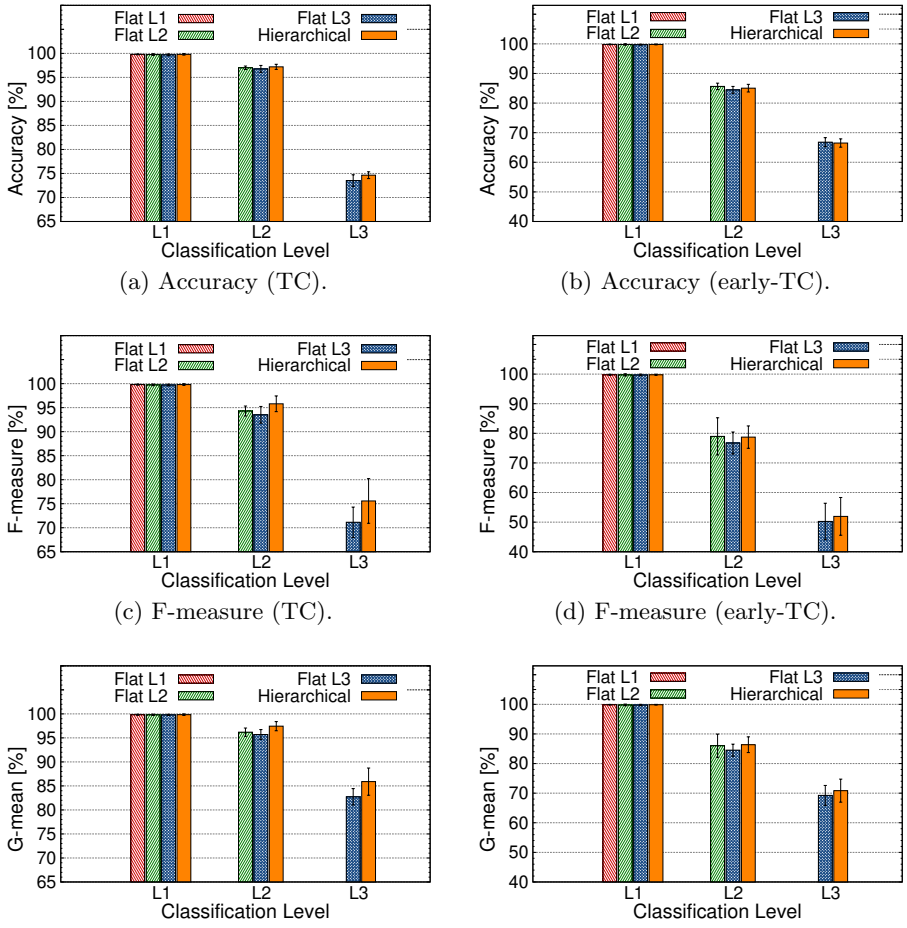
(g) L_1- Anonymous Network. (h) L_2- Traffic Type. (i) L_3- Application.

Figure 4.5: Accuracy (a–c), F-measure (d–f), and G-mean (g–i) [%] of hierarchical and flat classifiers: RF fed with different subsets of (statistical) features in `TC_set` (from 5 to 74 with increments of 5). Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

discriminating between 5 classes.

Impact of Feature Selection. As a first step towards the optimization of the proposed approach, we investigate here the performance of the framework when varying the number of features used by each classifier but *considering a common number of features (here denoted M) for each node in the tree*, and *keeping the classification algorithm fixed to RF*, being the best-performing one in the flat case also for different feature configurations as depicted in Fig. 4.5. We remark that although there is a common M (viz. number of features) for all the nodes, the specific set of features *may differ* depending on the related ranking.

Figure 4.5 summarizes the obtained results—with a view to finding the optimal value M *for the entire hierarchy*—also providing a comparison against the three flat classifier counterparts in terms of accuracy, F-measure, and G-mean. First, the results highlight that there is no appreciable performance difference among L_1–L_3 flat classifiers and the hierarchical approach by looking at L_1 metrics, and only a slight performance improvement is achieved with a high number of features. Also, performance saturation is observed with at least 10 features. On the other hand, at L_2 the hierarchical approach obtains slightly higher performance with a lower number of employed features per node (approximately 10–20), whereas at L_3 (being the harder TC task) the following key observations can be made: (i) the hierarchical approach provides a non-negligible improvement over the L_3 flat classifier for all the feature-range considered, (ii) the best performance of the HC is attained with a smaller number of features. These considerations apply to all the performance measures considered, with the most evident improvement attained by the HC framework in terms of F-measure.



(a) Accuracy (TC).

(b) Accuracy (early-TC).

(c) F-measure (TC).

(d) F-measure (early-TC).

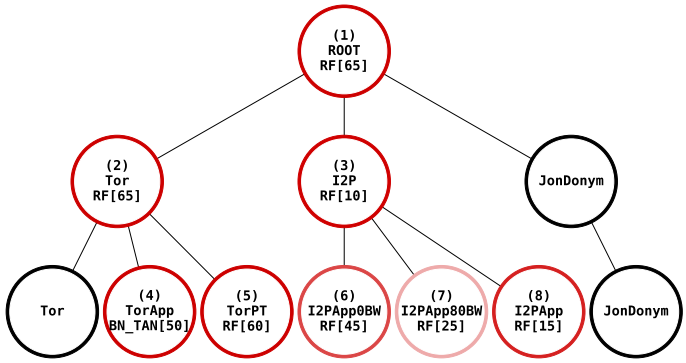
(e) G-mean (TC).

(f) G-mean (early-TC).

Figure 4.6: Accuracy (a), F-measure (c) and G-mean (e) of the best classifiers and of their early-TC counterparts (b, d, and f). Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

Fine-grained Optimization. Herein, the fine-grained framework optimization, in case of features from `TC_set`, is discussed. In detail, with the aim of trading off design complexity with performance, *we remove the constraints previously introduced, and allow each node in the hierarchy to be optimized in terms of both the number of features and the classifier type.* As remarked in Sec. 4.3.2, we consider *four* different ML-based classifiers, namely C4.5, RF, NB_SD and BN_TAN, these latter two being the best-performing Naïve Bayes and Bayesian Networks alternatives (cf. §4.4.1). We remark that, to avoid a combinatorial explosion of the optimization problem, we resort to the *per-node* optimization rationale described in Sec. 4.2.1. Based on the above rationale, in Fig. 4.6 we report the classification performance in terms of accuracy (Fig. 4.6a), F-measure (Fig. 4.6c) and G-mean (Fig. 4.6e), by comparing the flat classification approaches with the per-node optimized hierarchical classifier. This proposed (optimized) hierarchical classifier is able, at least, to perform at the t^{th} ($t = 1, 2, 3$) level on a par with the corresponding flat classifier *explicitly designed* to solve the classification task at the same level. In detail, such optimized hierarchical approach is able to achieve 99.81% (resp. 99.83%), 95.81% (resp. 97.44%) and 75.56% (resp. 85.89%) F-measure (resp. G-mean) score at L_1, L_2, and L_3, respectively. These results (almost) represent a tie at L_1, whereas +1.51% (resp. +1.73%) and +4.42% (resp. +3.16%) gains are experienced at L_2 and L_3, respectively. Moreover, from a statistical significance viewpoint, we observed a gain of the HC approach in 97.5% of the cases over the considered folds. The details of the optimized HC are reported in Fig. 4.7, where for each classifier node the employed classification algorithm and the number of features are reported. Remarkably, RF denotes the best classifier for each node-specific classification task, while only for the classifier of *Tor App* applications BN_TAN provides higher performance. Instead, the variability of

the optimal number of features underlines no clear trend, except that usually I2P-related node classifiers require a lower number of features, at least when leveraging those in `TC_set`.



(a) Fine-grained optimized hierarchical structure **with** `TC_set`.

Classifier	#Classes	Accuracy	F-measure	G-mean
$C_0 \rightarrow$ ①	$L_1 = 3$	99.81 ($\pm$ 0.06)	99.81 ($\pm$ 0.06)	99.83 ($\pm$ 0.05)
$C_1 \rightarrow$ ②	$\overline{L}_1 = 3$	99.97 ($\pm$ 0.04)	99.91 ($\pm$ 0.20)	99.97 ($\pm$ 0.04)
$C_2 \rightarrow$ ③	$\overline{L}_2 = 3$	95.05 ($\pm$ 0.29)	91.53 ($\pm$ 1.06)	93.29 ($\pm$ 0.79)
$C_{11} \rightarrow$ ④	$\overline{L}_{11} = 3$	99.58 ($\pm$ 1.25)	99.58 ($\pm$ 1.25)	99.69 ($\pm$ 0.94)
$C_{12} \rightarrow$ ⑤	$\overline{L}_{12} = 5$	99.72 ($\pm$ 0.12)	99.44 ($\pm$ 0.21)	99.63 ($\pm$ 0.14)
$C_{21} \rightarrow$ ⑥	$\overline{L}_{21} = 3$	72.42 ($\pm$ 1.32)	58.43 ($\pm$ 1.63)	69.00 ($\pm$ 1.26)
$C_{22} \rightarrow$ ⑦	$\overline{L}_{22} = 3$	48.94 ($\pm$ 0.51)	48.90 ($\pm$ 0.52)	60.37 ($\pm$ 0.42)
$C_{23} \rightarrow$ ⑧	$\overline{L}_{23} = 5$	75.12 ($\pm$ 5.95)	72.50 ($\pm$ 6.99)	81.68 ($\pm$ 4.58)

(b) Performance metrics **with** `TC_set`.

Figure 4.7: Fine-grained optimized hierarchical structure with `TC_set` (a). Optimal number of features for each classifier node is shown in square brackets. Lighter red color points to worse performance. Related per-node metrics are shown in (b), with $< 60\%$ F-measure nodes highlighted in gray.

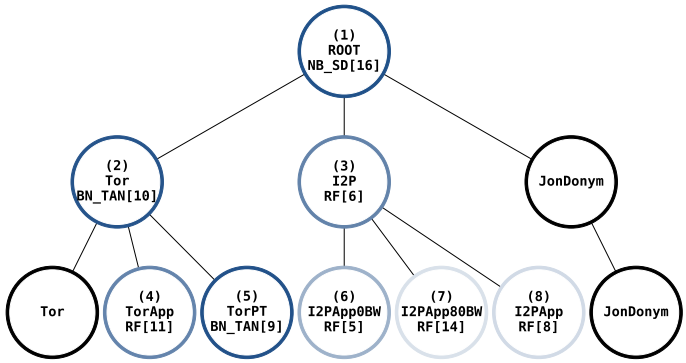
Optimization for Early Traffic Classification. Herein we evaluate the hierarchical framework when the classifiers are fed with features in `EarlyTC_set` (see §4.3.2) to investigate the possibility of performing "early"

classification of anonymous traffic. For the present analysis, we consider PLs and IATs of the first $K = 1, \ldots, 16$ (non-zero payload) packets. Specifically, we remove from dataset D_5 (see Fig. 2.9) all the instances whose first $K = 20$ packets have *all zero payload*. We recall that these correspond to a super-set of those removed in the case of flow-based TC (*i.e.* the resulting dataset $\bar{D}_5$), as we are removing also the instances with some payload exchange after the first 20 (zero-payload) packets. The reason for a different filtering procedure is to avoid submitting "non-informative" (referring to the first $K = 20$ packets) instances to the considered classifiers.

This analysis helps assessing the framework capability in supporting early TC, that is to evaluate how soon and to which degree ATs and related services can be identified. To this end, we have paralleled the previous investigations in the early-TC scenario. Nonetheless, herein we omit the redundant details, and comment only the final (most interesting) results. We remark again that the main difference with the previous analysis concerns the *feature selection* process, herein performed on a *time-basis* (*i.e.* only the features drawn from the first K packets are considered).

First, the results show that a naïve hierarchical "extension" of the best flat L_3 classifier (in this case a BN_TAN with $K = 11$ packets) is *not* able to provide improved performance (*e.g.*, 48.80% F-measure at L_3, as opposed to 50.23% in the flat case). This result highlights a key difference with respect to the scenario leveraging `TC_set` and emphasizes the *need for hierarchical-specific optimization* in such case. Secondly, we investigate (as in Fig. 4.5) how varying the features corresponding to the first K packets could improve the performance of the naïve hierarchical extension, and compare it with the best flat counterpart. Our investigations reveal that a performance saturation is observed after $\approx$ 10 packets, and that the HC approach is able to improve the best flat L_3 approach in terms of G-mean, whereas

a performance drop both in terms of accuracy and F-measure is observed for all the values of K considered. Finally, fine-grained optimization (see Figs. 4.6b, 4.6d, and 4.6f) provides higher performance with respect to the optimized flat case, *e.g.*, +1.71% F-measure and +1.59% G-mean at L_3. In this case, from a statistical significance viewpoint, we observed a gain of HC approach in 90.0% of the cases over the considered folds.



(a) Fine-grained optimized hierarchical structure **with** `EarlyTC_set`.

Classifier	**#Classes**	**Accuracy**	**F-measure**	**G-mean**
$C_0 \rightarrow$ ①	$L_1 = 3$	99.80 (± 0.05)	99.78 (± 0.06)	99.87 (± 0.04)
$C_1 \rightarrow$ ②	$\overline{L}_1 = 3$	99.20 (± 0.12)	90.48 (± 1.61)	94.82 (± 1.27)
$C_2 \rightarrow$ ③	$\overline{L}_2 = 3$	72.20 (± 0.81)	60.61 (± 1.70)	66.85 (± 1.02)
$C_{11} \rightarrow$ ④	$\overline{L}_{11} = 3$	63.42 (± 4.92)	63.27 (± 4.92)	72.01 (± 3.85)
$C_{12} \rightarrow$ ⑤	$\overline{L}_{12} = 5$	99.69 (± 0.07)	99.55 (± 0.18)	99.63 (± 0.13)
$C_{21} \rightarrow$ ⑥	$\overline{L}_{21} = 3$	67.37 (± 0.81)	44.56 (± 1.45)	56.38 (± 0.97)
$C_{22} \rightarrow$ ⑦	$\overline{L}_{22} = 3$	43.47 (± 0.75)	43.13 (± 0.72)	55.76 (± 0.63)
$C_{23} \rightarrow$ ⑧	$\overline{L}_{23} = 5$	50.67 (± 9.21)	37.75 (± 11.24)	58.04 (± 7.81)

(b) Performance metrics with **with** `EarlyTC_set`.

Figure 4.8: Fine-grained optimized hierarchical structure with `EarlyTC_set` (a). Optimal number of features for each classifier node is shown in square brackets. Lighter blue color points to worse performance. Related per-node metrics are shown in (b), with $< 60\%$ F-measure nodes highlighted in gray.

The corresponding optimized hierarchical structure is shown in Fig. 4.8 and—when compared to Fig. 4.7—clearly shows that per-node optimized classifiers significantly differ in type and number of features, thus motivating the need for fine-tuned optimization of the proposed HC. Although the HC gain is not significant, its operating principle allows to delve into the "information structure" of the anonymous TC problem and highlight critical points, by excluding potential performance drops due to the size of the classification task, as shown by the following detailed performance analysis.

4.4.3 Detailed Traffic Classification Performance

In this section, we perform a detailed performance evaluation of HC, discussing the results of per-node and per-class breakdowns and, finally, the application of a censoring threshold to flat and hierarchical ATs' traffic classifiers.

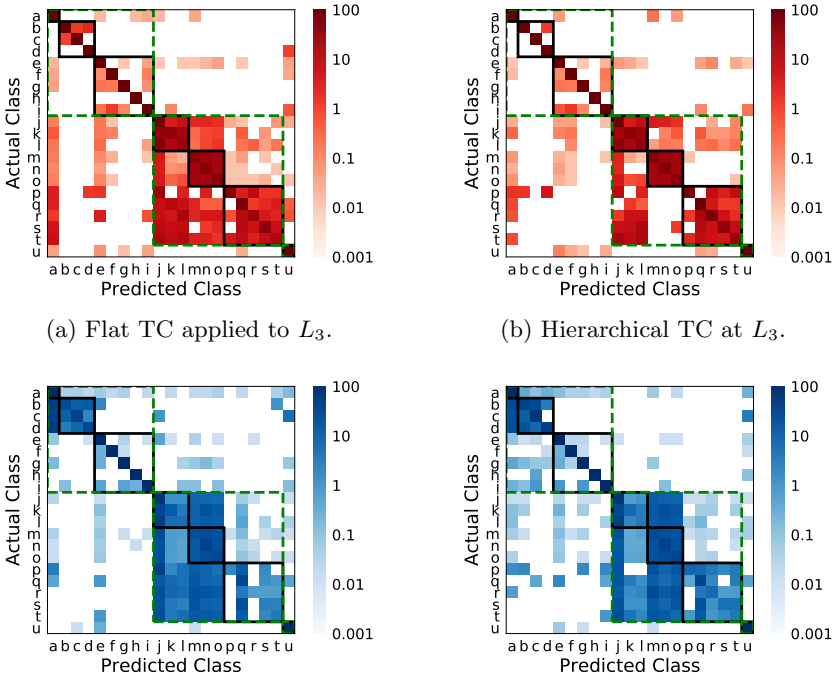
Per-node Detailed Classification Performance. We report in Tabs. 4.7b and 4.8b the detailed per-node classification performance, corresponding to our optimized hierarchical classifier fed with `TC_set` and `EarlyTC_set` features, respectively.

The following interesting observations can be made on the reported tree representation. First, with respect to *Anonymous Network Level* classification (L_1), the nodes present near-ideal performance both when relying on `TC_set` and `EarlyTC_set`, thus showing *almost no errors propagating from HC of anonymous networks*. Secondly, at *Traffic Type Level* (L_2) both the approaches based on `TC_set` and `EarlyTC_set` show near-ideal performance in classifying *Tor* traffic types, whereas some performance degradation is observed in classification of I2P traffic types; this phenomenon is more penalizing in the early-TC case, with *I2P* F-measure dropping down to 60.61%.

This represents one of the main causes of performance difference among the two scenarios, as all these errors are propagated downwards. Finally, at *Application Level* (L_3), the behavior of the classifier nodes is more varied. Indeed, referring to the approach based on `TC_set`, it is apparent that Tor nodes (*i.e. TorApp* and *TorPT*) work well, whereas on I2P nodes significant degradations (higher than those at L_2) are observed, with *I2PApp80BW* suffering the most significant. Differently, discrimination within each I2P traffic type and *TorApp* is only possible with $< 65\%$ F-measure with the features in `EarlyTC_set`. Therefore, classification within *I2PApp80BW* cannot be accurately attained with neither of the considered feature sets, thus confirming the intuition that *I2PApp80BW* represents traffic obtained by mixing different apps. In such a case, the advantage of the classification task split into sub-problems cannot exceed a certain threshold, due to the impossibility of discerning applications within this traffic type, resorting to the set of the available features, suggesting the use of more sophisticated classifier nodes (as DL-based ones described in Chapter 5) and HC approaches.

Nevertheless, this analysis witnesses the appeal and effectiveness of the HC framework also in real environments and provides insights in identifying performance bottlenecks which lie on a very limited set of nodes in the hierarchy (*e.g.*, those in charge of classifying applications running within I2P). This, for example, confirms the common thinking that I2P provides a higher privacy level, from the TC viewpoint, with respect to Tor, at the expenses of increased latency.

Per-Class Performance Breakdown. The classification at L_3 is a challenging task but also the most interesting from a user's privacy perspective. Consequently, in Fig. 4.9 we report the per-class performance breakdown of HC results in terms of confusion matrices at L_3, comparing them against



(a) Flat TC applied to L_3.

(b) Hierarchical TC at L_3.

(c) Flat early TC applied to L_3.

(d) Hierarchical early TC at L_3.

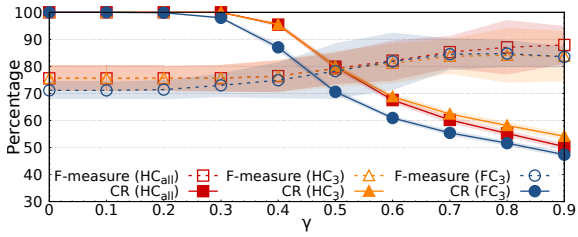
Figure 4.9: L_3 confusion matrices ([%] in log scale) of the best flat and hierarchical classifiers in flow-based TC (red) and early TC (blue).

the results obtained with the best-performing flat approach, being the RF fed with 65 statistical features and the BNs fed with the first 11 packet lengths in the case of flow-based and early TC, respectively. We recall that for these matrices the higher the concentration toward the main diagonal, the better the overall performance. Further, to highlight how errors at L_3 which do not imply misclassifications at L_1/L_2 are less severe and should be promoted as opposed to the others, in the same figures we also highlight the error patterns confined to the same traffic type (solid boxes) and the

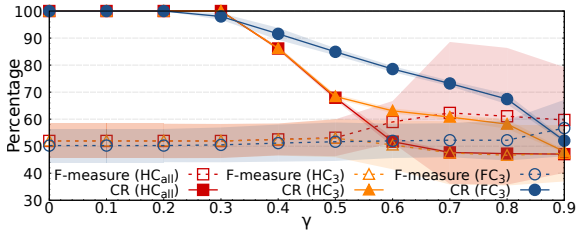
same anonymous network (dashed boxes).

First, comparing flat and hierarchical approaches (Figs. 4.9a and 4.9b, respectively), we can observe a reduction of error-patterns in the latter case, highlighting the beneficial "divide-et-impera" principle of HC. Secondly, confusion matrices at L_1 (dashed boxes) show that classifiers based on both approaches (with different quantitative outcomes) present error patterns which almost entirely lead to a misclassification of the traffic type within the same anonymous network. Differently, referring to L_2 standpoint (solid boxes), HC provides improved capabilities in confining errors to the same traffic type, especially in the case of I2P traffic. A similar consideration applies to early-TC results (Figs. 4.9c and 4.9d) and, in particular, to the errors concerning the applications of *I2P Apps* and *Tor Apps*. Hence, HC approach performance clearly highlights the inability, not depending on the size of the classification task, in satisfactorily discriminating (with an early-TC setup in mind) among I2P traffic types and within their corresponding applications, along with those in *Tor Apps*. On the other hand, the results confirm the outcomes of [160], witnessing that the obfuscation implemented by *Tor Pluggable Transports* induces a class fingerprint easily distinguishable ($\geq$ 99% accuracy, see C_2 classifier in Figs. 4.7 and 4.8) from both *Normal Tor Traffic* and *Tor Apps*. Overall, this evaluation, delving into misclassification patterns and their severity, proves that the HC approach better confines misclassifications in the same anonymous network and even in the same traffic type.

Performance with Reject Option. Finally, in Fig. 4.10 we focus on the adoption of censoring threshold(s) for flat and hierarchical classification of ATs. Specifically, we report L_3 performance, being the hardest task considered. We recall that, although in the flat case there is only a single



(a) TC.



(b) Early TC.

Figure 4.10: F-measures and Classified Ratios (CR) of best classifiers vs. γ.

tunable γ (referred to as FC_3), HC allows to set a different threshold value at each node (*e.g.*, for C_{ij} the threshold γ_{ij} can be adjusted independently from the others, see Sec. 4.2.1). Nonetheless, as a preliminary investigation toward the censored behavior of the HC approach—to avoid cumbersome analyses—we consider two simplified options: considering only a common γ value shared by (a) *all the nodes* in the hierarchy (HC_{all}) and (b) *solely* by the classifier nodes concurring to L_3 classification (HC_3). Accordingly, we report the F-measure vs. γ (similar trends have been observed for the other metrics) along with the corresponding trend of the Classified Ratio (CR) (viz. the percentage of classified samples).

Although using such thresholds cannot be intended as a *panacea* able to cope with high-confidence wrong predictions, results pertaining to the adoption of `TC_set` show performance improvement with increasing γ. In

detail, both hierarchical variants offer performance gain with higher CR—corresponding to less discarded flows—with respect to FC_3 (*e.g.*, $\approx 80\%$ F-measure is attained with $\approx 80\%$ CR), with HC_{all} having slightly improved performance, while incurring in a slightly lower CR, due to the presence of non-zero thresholds for nodes at higher levels in the hierarchy. On the other hand, in the early-TC scenario, the CR is lower for hierarchical approaches, while HC_{all} provides slightly improved performance than FC_3, whose performance do not benefit from a γ increase. Nonetheless, such results underline how progressive censoring via per-node thresholds may be a viable option for further performance improvement and paves the way for developing advanced HC optimization, based, for example, on the automatic computation of reject thresholds possibly differing at each node.

Chapter 5

Traffic Classification using Deep Learning

In Chapters 3 and 4, we have described two different "structural" (viz. design) enhancements (*i.e.* multi- and hierarchical-classification approach, respectively) to standard ML traffic classifiers, able to improve classification performance at various degrees. In addition to these latter, in this chapter we propose the adoption of Deep Learning (DL) paradigm for the design of classifiers able to deal with the challenging characteristics of mobile and encrypted traffic. Indeed, the successful use of ML classifiers, both standard and based on more complex structures (as MC and HC), relies on obtaining handcrafted features from traffic objects that are suitable to accomplish the specific classification task. As seen in previous chapters, in the TC context, these usually correspond to statistics or other information (see Tabs. 3.1 and 4.1) extracted from the sequence of PLs or IATs.

On the other hand, in Sec. 1.3.4, we have shown how DL allows training classifiers directly from input data by automatically distilling structured and complex feature representations [68], avoiding the domain-expert driven feature-design process peculiar of "traditional" ML-based classification, be-

ing usually time-consuming, hardly automatable, and unable to keep the pace of (mobile) network traffic evolution and mix. Since this process might preclude the design of accurate and up-to-date mobile-traffic classifiers, we believe that DL may be the stepping stone toward high-performing TC in the dynamic and demanding mobile context, marked by a high number of apps, possibly generating similar traffic patterns and with complex fingerprints due to scarce number of training samples per app and device/OS/version diversity. This intuition is also confirmed by recent survey [28] discussing the application of DL to mobile and wireless networking that identifies in advanced DL techniques a powerful enabler for effective app-level data mining in encrypted TC.

However, DL benefits should not be taken for granted and its naïve adoption to (mobile and encrypted) TC may imply misleading design choices and lead to biased conclusions, due to the peculiar (and tricky) nature of network traffic data. Last but not least, this traffic-data nature is *heterogeneous* and its whole capitalization is yet to be achieved. This constitutes, in our opinion, one of the main gaps to fill (viz. the prerequisite) for the successful employment of DL assets to mobile TC, thus echoing its fruitful use in "mature" fields, such as image and natural language processing [68].

Hence, we propose for the first time (cf. Tab. 5.1) the *design of mobile traffic classifiers able to operate with encrypted traffic via the adoption of DL umbrella*. To this end, we resort to the development of a systematic framework for the design of novel DL-based TC architectures and comparison of existing ones, declined in this chapter in the mobile scenario, but having a wider applicability to encrypted TC. This originates from a critical analysis (later provided in Sec. 5.1) of several non-mobile-specific DL classifiers recently appeared in TC literature [21, 47, 81, 91, 92, 94] and here reproduced, so as to avoid focusing on a specific DL technique and draw

close-to-general conclusions. In detail, the proposed framework dissects the DL-based TC problem from different viewpoints (highlighted via Fig. 5.1): (A) the traffic object adopted, (B) the type (and the amount) of input data fed to the DL classifier, (C) the DL architecture employed, and (D) the required set of performance measures for an objective and comprehensive evaluation.

On the basis of these design milestones, we resort to the proposed framework to provide a constructive design-oriented contribution for addressing the multi-view capitalization of traffic data via a novel *multi-modal DL-based mobile traffic classification (MIMETIC)* architecture, having the capability of exploiting effectively the heterogeneous nature of the different views of a traffic object (*e.g.*, payload bytes or header fields), by capturing both intra- and inter-modalities dependence. It is worth noting that multi-modal DL constitutes also a full-fledged generalization of the multi-classification framework—in this context also referred to as late (or score/decision) fusion—devised in Chapter 3. Although its adoption is obtaining a growing and wider interest in the scientific literature [164, 165, 166], *no such approach has been proposed in (mobile) TC literature to date*, up to our knowledge. Since the capitalization of multi-modality in DL architectures is far from trivial [166], it requires a thorough design which cannot ignore expertise from network traffic monitoring. Hence, MIMETIC approach is carefully defined herein in terms of the general architecture and proposed training procedure.

Our framework (expressly the evaluation workbench) is then applied to a realistic experimental setup, consisting of the three different mobile datasets of real human users' activity described in Sec. 2.5.1. Our aim is to assess the most appealing (single-modal) DL techniques, the potential gain with respect to ML-based best alternatives and shallow architectures—

so as to justify the need for complex hierarchically-arranged features—and highlight open issues for real-time and accurate mobile TC via DL. Up to our knowledge, *no similar systematic approach and experimental investigation have been performed in the mobile scenario to date.*

Moreover, we evaluate MIMETIC performance in the same setup and compare it with the best (resulting from the aforementioned assessment) single-modal DL-based and state-of-the-art ML-based traffic classifiers previously dissected, so as to draw close-to-general take-aways. Experimental results highlight a performance improvement of the implemented MIMETIC instance in terms of both concise and fine-grained measures, while reporting a lower training time (*more than three times*) with respect to existing (single-modal) DL-based traffic classifiers. Specifically, the proposed implementation *outperforms the best baseline* up to +8.66% in terms of F-measure (*i.e.* 82.99% when classifying the traffic generated by iOS apps). The improvement is also observed with respect to classifier fusion (as described in Chapter 3) of best single-modal DL baselines, also unexplored to date. To provide a finer performance control, we also enrich MIMETIC (and baselines) with the option of censoring some "unsure" classifications (*i.e.* it is equipped with a "reject option", see §1.3.2). Corresponding results report very high performance with a moderate (controllable) number of unclassified instances. Lastly, we deepen their (soft-output) behavior via a calibration analysis to verify if the estimated class-prediction probabilities are representative of the actual class probabilities.

The outcomes of this analysis underline the deficiencies of current DL-based traffic classifiers and the need for: (*i*) unbiased, informative, and heterogeneous inputs extrapolated from traffic data, (*ii*) sophisticated (*e.g.*, multi-modal) DL architectures, and (*iii*) a rigorous and multifaceted performance evaluation. In this regard, it represents a first attempt to ad-

dress (*i*) and (*ii*) issues, being also a "safe" groundwork for paving the way to the design of accurate DL-based classifiers coping with highly-diverse mobile traffic and able to capitalize its multi-view nature, whereas it provides designers with a fine-level performance evaluation workbench (*iii*).

The rest of this chapter is organized as follows. Sec. 5.1 reviews the related DL-based TC literature; Sec. 5.2 describes the DL framework for mobile TC, focusing on key aspects to address, including the performance evaluation workbench here proposed; in Sec. 5.3, we illustrate the MIMETIC architecture, the training procedure adopted, and its implemented instance; finally, the experimental evaluation is reported and discussed in Sec. 5.4.

5.1 Related Works

In this section, we firstly provide an intuitive categorization, via a systematic taxonomy, of literature on DL-based Internet TC. Successively, we give the details of these works, briefly discussing also DL applied to the conceptually-similar task of WF, along with a wrap-up discussion that highlights the limitations of current literature.

As pointed out in Secs. 1.4 and 3.1, a number of works have faced mobile TC in the last five years, under encrypted-traffic assumption, mostly using ML and based on bot-generated traffic (see Tab 1.2). On the other hand, the appeal of DL to TC is confirmed by several recent works providing initial design attempts of DL-based traffic classifiers, either not-mobile or not-encrypted. In Tab. 1.2, we have already shown that all these works use human-generated traffic datasets to evaluate their proposals. Also, from our thorough search, TC in the mobile and encrypted scenario by means of DL appears unexplored to date.

In detail, in Tab. 5.1, we summarize and categorize each work performing

Table 5.1: Summary of previous works (by year) tackling TC-related tasks via DL-based approaches. Only our *DL framework & MIRAGE* deal with mobile and encrypted app traffic using a multimodal DL-based classifier.

Paper	Task	MT	ET	TO	Classes$^{\diamond}$	Input Data	F	Classifier	MM	Performance
Wang [81]	TI/TC	○	○	BF	25 protocols	TCP payload [1000 B]	○	SAE	○	$\geq$ 90% prec. & rec.
Wang *et al.* [21]	TC	○	●	F/BF	$\leq$ 20 malware/apps	PCAP/L4 payload [784 B]	○	2D-CNN	○	$\geq$ 89% per-class metrics
Wang *et al.* [94]	TI/TC	○	●	F/BF	$\leq$ 12 traffic types	PCAP/L4 payload [784 B]	○	1D-CNN	○	+2.51% w.r.t. [21]
Chen *et al.* [90]	TC	○	●	BF	5 protocols/apps	PS, IAT, PD [10 packets]	○	2D-CNN	○	88.4% best acc.
Lotfollahi *et al.* [92]	TC	○	●	P	17 / 12 apps	L2 payload [1500 B]	○	SAE, 1D-CNN	○	95% / 97% F-meas.
Lopez *et al.* [91]	TI/TC	○	●	BF	108 services	6 fields [20 packets]	○	LSTM+2D-CNN	○	95.7% best F-meas.
Rimmer *et al.* [25]	WF	○	●	TS	900 websites	Cell directions [150÷5k]	○	SDAE, CNN, LSTM	○	94% acc.
Oh *et al.* [47]	WF	○	●	TS	100 websites	Cell directions [784]	○	MLP, 2D-CNN, AE	○	92% / 1% rec. / fall-out
Vu *et al.* [93]	TI	○	●	BF	2 SSH/nonSSH	Flow-based statistics	●	AC-GAN	○	$\approx$ 95% F-meas.
Li *et al.* [89]	TC	●	○	H	12 Android apps	HTTP fields [28×36 B]	○	VAE	○	99.6% acc.
Chen *et al.* [96]	TC	○	●	P	6 apps	L2 payload [39×39 B]	○	2D-CNN	○	> 85% acc.
Li *et al.* [101]	TC	○	●	P	10 protocols/apps	L4 payload	○	LSTM, bi-GRU	○	$\geq$ 90% F-meas.
Huang *et al.* [23]	TI/TC	○	●	BF	9 Trojans	PCAP [1024 B]	○	2D-CNN	○	> 90% per-class metrics
Shi *et al.* [98]	TC	○	●	BF	10 traffic types	ML&DL-selected features	●	DBN	○	$\approx$ 60% G-mean
Sirinam *et al.* [26]	WF	○	●	TS	100 websites	Cell directions [5k]	○	SDAE, CNN	○	99% / 94% prec. / rec.
Zhang *et al.* [99]	TC	○	●	BF	10 services	Flow-based statistics	●	SAE	○	$\geq$ 90% F-meas.
Wang *et al.* [100]	TC	○	●	P	15 apps	L2 payload [1480 B]	○	MLP, SAE, 2D-CNN	○	$\geq$ 96% F-meas.
Liu *et al.* [102]	TC	○	●	BF	18 apps	IP packet lengths [128]	○	bi-GRU	○	$\approx$ 99% mean TPR
Sun *et al.* [103]	TC	○	●	BF	$\leq$ 50 apps	Flow-based statistics	●	DNN	○	$\approx$ 97 best acc.
Zeng *et al.* [104]	TI/TC	○	●	BF	6 traffic types	PCAP [900 B]*	○	1D-CNN, LSTM, SAE	○	99.8% best acc.
DL framework & MIMETIC	*TC*	●	●	*BF*	49 *Android apps* 45 *iOS apps*	*PCAP/L4 payload [256÷2304 B] 4 - 6 fields [4÷32 packets] PD [784 packets]*	○	*SAE, LSTM, GRU, 1D-CNN, 2D-CNN, LSTM+2D-CNN*	●	+8.6% *MIMETIC acc. w.r.t. best baseline*

Multi-Modal (MM). **Encrypted Traffic (ET)**. **Mobile Traffic (MT)**. **Features (F)**.
Task: Traffic Classification (TC), Traffic Identification (TI), Website Fingerprinting (WF).
Traffic Object (TO): biflow (BF), flow (F), HTTP session (H), packet (P), Tor cell sequence (TS).
Input Data: inter-arrival times (IAT), packet directions (PD), packet sizes (PS), raw data of PCAP trace (PCAP), X^{th} layer of ISO/OSI model (LX).
Classifier: Auxiliary Classifier Generative Adversarial Network (AC-GAN), AutoEncoder (AE), Bidirectional Gated Recurrent Unit (bi-GRU), Convolutional Neural Network (CNN), Deep Belief Network (DBN), Deep Neural Network (DNN), Long Short-Term Memory (LSTM), MultiLayer Perceptron (MLP), Stacked AutoEncoder (SAE), Stacked Denoising AutoEncoder (SDAE), Variational AutoEncoder (VAE); + symbol indicates hybrid architectures.
$\diamond$: in **Classes** "apps" do not refer to mobile apps, if not stated explicitly (*e.g.*, Android or iOS).
*: TCP/UDP headers and MAC addresses are removed.

TC-related tasks via DL based on (*a*) the specific task it tackles (*i.e.* TI, TC, or WF), whether (*b*) it focuses on the mobile scenario, and (*c*) it tackles encrypted TC. For each study, we surface from a design viewpoint: (*i*) the traffic segmentation criterion employed (*i.e.* the *traffic object*), (*ii*) the classes to discriminate between, (*iii*) the input type used to feed the classifier, (*iv*) the specific DL classifier adopted, and (*v*) whether the DL architecture is multi-modal (*i.e.* it is fed with multiple types of input). Furthermore, the flag *features* integrates (*iii*), stressing the counter-productive use of handcrafted features as input data for DL architectures. Finally, we report the (best) classification performance each proposal achieves. The above categorization prompts some caveats and *warning flags* in the adoption of the approaches reported in Tab. 5.1 to the mobile and encrypted context. Each of these is discussed hereinafter with regards to each separate aspect.

Regarding the *traffic objects*, we observe that the flows and biflows are the most-common choices under the encrypted-traffic assumption, whereas the HTTP sessions cannot be used in presence of encrypted traffic, due to the need to access the cleartext of transport layer payload to define such packet aggregation. Similarly, though DL-based TC can in principle be performed on a per-packet basis [92], the common labeling among packets of the same communication and the unavailability of cleartext payload in each encrypted packet discourage the use of this traffic object.

Regarding *inputs*, although raw payload is widely used as a relevant input type for DL architectures, the size and layer chosen vary from work to work and layer choices lower than transport level are likely to introduce bias in TC performance as we shall demonstrate in Sec. 5.4.1. The same reasoning applies to byte-converted raw traces including also PCAP metadata [21, 94] and inputs comprising source/destination port fields [91]. Equally important, the counter-productive application of DL to manually-

extracted traffic features, as opposed to input data, nullifies a key asset of DL paradigm, that is no need of human-expert intervention for designing informative features.

Referring to *DL architectures* all the works have designed DL traffic classifiers based on a single input type (viz. single-modal). Furthermore, some research [21, 23, 96, 90, 91, 100] has used arbitrarily-shaped 2-D convolutional layers as the relevant block to handle a naturally 1-D input (*i.e.* a traffic packet series). Lastly, only the work in [91] (in addition to our *DL framework*) started exploiting the composition possibilities offered by hybrid architectures (marked with + in the *classifier* column) allowed by the connectionist philosophy underlying DL.

Details on State-of-the-art Traffic Classification via Deep Learning. Henceforth, we first discuss the applications of DL to the problem of encrypted WF, then we complete our review of related literature by analyzing recent DL proposals to standard TC. Oh *et al.* [47] study the usage of DL for WF and also prove its effectiveness on feature extraction—via an AutoEncoder (AE)—for state-of-the-art ML algorithms. The results underline that DL architectures successfully detect which website the user visited among 100 ones against 100k background websites. A novel DL-based method to deanonymize Tor traffic is proposed in [25] and tested on a very-large WF dataset made of $\geq 3 \cdot 10^6$ network traces. The results highlight that the performance achieved via DL is comparable to state-of-the-art deanonymization attacks, with the best-performing DL model being +2% accurate. Finally, Sirinam *et al.* [26] develop a WF attack against Tor which is evaluated against state-of-the-art defenses (*i.e.* WTF-PAD and Walkie-Talkie). Performance evaluation in an open-world setting shows that the attack is effective against undefended traffic, while still relevant (95/70% of

precision/recall) in case WTF-PAD defense is employed.

A first DL approach applied to cleartext TI and TC, but seamlessly applicable also to encrypted traffic, is presented in [81], employing Stacked AutoEncoders (SAEs) and comparing them to standard neural networks. The results show that the SAE outperforms the latter and achieves $\geq 90\%$ precision and recall in protocol identification (on 25 most popular protocols), and $\geq 80\%$ class prediction probability on 6.7k out of 10k traffic samples unrecognizable via DPI. The SAE, although trained on manually-designed features, is also recently applied to TC in [99], showing that it outperforms an SVC and achieves a high F-measure ($\geq 90\%$) on a real-world dataset comprising the traffic of 10 different services.

On the other hand, Wang *et al.* [21] propose a novel malware TC, based on 2D-Convolutional Neural Networks (CNNs) and explicitly devised for encrypted traffic. The approach is tested on a dataset ($\approx$ 752k instances) consisting of (*i*) 10 malware traffic types from public websites and (*ii*) 10 normal traffic types, in two different tasks: (*i*) malware vs. normal (TI) and (*ii*) traffic-type (20 classes) classification. Also, to feed the classifier the authors use two different choices of raw "traffic images" (named "ALL" and "L7") dependent on the protocol layers considered to extract the input data, showing that biflow-based TC with "ALL" (referred to as "PCAP" in Tab. 5.1) is the most informative and reaches the best performance for all the metrics considered. Unfortunately, such design choice led to biased results[1]. In [94] the same authors devise a similar approach for encrypted TC based on the 1D-CNN. The experiments are conducted on a selection of the "ISCX VPN-nonVPN" (non-mobile) dataset [107] and consist of four different setups including VPN/nonVPN TI, encrypted TC (6 or 12 classes), and

[1]We provide a thorough comparison taking into account this issue in Sec. 5.4.1.

TC of VPN-encapsulated data (6 classes). Consistently with [21], the configuration "Biflow + ALL" performs the best. Moreover, the configuration-optimized 1D-CNN always achieves higher accuracy than a 2D-CNN counterpart (being both however $\geq 80\%$) in all the setups, uniformly with the 1-D nature of traffic packet series, and almost always outperforms the C4.5 classifier originally designed in [107]. The same dataset is used to test *Deep Packet* [92] and *Datanet* [100], two DL-based encrypted traffic classifiers working at packet-level and adopting a 1D/2D-CNN, a (deep) MultiLayer Perceptron (MLP), or a SAE. In the former case, Deep Packet achieves an average 95% (resp. 97%) F-measure for the application identification (resp. traffic characterization) task, consisting of 17 applications (resp. 12 activities). In the latter case, Datanet reaches $\geq 96\%$ F-measure with both the SAE and 2D-CNN in discriminating among a subset of 15 applications. In both studies the first 1480 bytes of L2 payload are used as the input, thus leading to biased performance. Deep-Full-Range [104] is another framework leveraging DL for encrypted TC and intrusion detection that is evaluated by means of the "ISCX VPN-nonVPN" dataset along with the "ISCX 2012 IDS" one [167]. An 1D-CNN, Long Short-Term Memory (LSTM), and SAE with L1 regularization are adopted for both the tasks on two subsets of the above-mentioned datasets, comprising 6 encrypted and 5 malware traffic types, respectively. The best performance is obtained with an 1D-CNN in the first case (99.8% accuracy) and an LSTM in the latter (99.4% accuracy).

Different DL architectures for encrypted TC, based on hybrid compositions of LSTM and 2D-CNN layers, are proposed in [91]. The best-performing of these variants attains an accuracy (resp. F-measure) up to 96.32% (resp. 95.74%) on a dataset captured on the Spanish academic backbone network and consisting of $\approx$ 266k biflows belonging to 108 distinct services. The analysis also highlights (i) a performance drop by including IATs

in the input and (*ii*) that $5 \div 15$ packets are enough for satisfying results. LSTM and Gated Recurrent Unit (GRU) layers are also employed in the *Byte Segment Neural Network* architecture [101] proposed for datagram-based classification and based on L4 payload. The experimental analysis, on a self-generated dataset made of 10 classes (protocols and applications), reports a $\geq 90\%$ F-measure for the applications/protocols considered. As a further innovation, Huang *et al.* [23] propose a multi-task DL approach (with a 2D-CNN) to simultaneously solve TI and TC tasks: (*i*) malware binary detection, (*ii*) binary recognition of VPN-encapsulation, and (*iii*) Trojan classification (9 classes). Devised approach is successfully tested on data assembled from "CTU-13" malware and "ISCX VPN-nonVPN" traffic datasets.

Similarly, Chen *et al.* [90] propose *Seq2Img*, a pipeline made of reproducing kernel Hilbert space embeddings (producing an equivalent image) and a 2D-CNN architecture, suitable for early TC (namely, based on the first 10 packets), where three packet informative fields (*i.e.* the PL difference, IAT, and packet direction) and the server IP address are used as the input. The approach is validated on two self-generated datasets whose traffic is related to five protocols and five Internet applications, respectively, achieving 99.84% and 88.42% accuracy. Shi *et al.* [98] devise a novel feature optimization approach, based on deep belief networks and ML-based feature selection techniques to improve TC performance, by overcoming the negative impacts of multi-class imbalance and concept drift. Experiments on real traffic traces show that the proposed approach outperforms existing ML classifiers and a deep belief network without feature selection. Another application of DL to TC with imbalanced network data is found in [93], where an auxiliary-classifier generative adversarial network is used to generate synthesized samples again in the form of a set of handcrafted features,

for training set balancing, to be used by ML classifiers. The method, tested on the NIMS dataset [136], outperforms a counterpart based on the synthetic minority over-sampling technique.

More recently, Sun *et al.* [103] also employ a set of 16 handcrafted statistics for feeding a multi-output deep neural network to simultaneously solve three classification tasks, namely the duration, the flow rate (both divided in two classes based on their median values), and the application. The authors leverage four subsets of two network traffic traces dataset (*i.e.* the WITS [168] and MOORE [106] ones) to evaluate their approach. Experimental results attain up 95.82%, 98.17%, and 97.26% accuracy for duration, flow rate, and application classification, respectively. However, performance degradation is evidenced when transfer learning (*i.e.* training the classifier on the target task starting from a structure pre-trained using data from the other tasks) or one shot learning (*i.e.* training the classifier with a lot of data from the other tasks and small data from the target task) are taken into account.

FS-Net is a model for encrypted TC proposed in [102] that jointly learns representative features from the packet length sequences of each biflow and uses them for classification. It leverages a multi-layer bidirectional-GRU encoder to learn the representation of the sequence and a multi-layer bidirectional-GRU decoder to reconstruct the original sequence, using both encoded and decoded features for TC. Authors uses a dataset self-captured from a real-world campus network environment [169], encompassing $>$ 956k biflows from 18 applications, to evaluate FS-Net. Comparison with state-of-the-art feature generation and TC techniques shows that FS-Net outperforms them reaching up to 99% average TPR against 0.05% average FPR.

To the best of our knowledge, the sole application of DL to mobile TC seems to be [89], where a DL classifier, based on variational AE and input

data taken from the reconstructed HTTP session (*i.e.* designed only for *clear traffic*) is proposed and tested on a self-generated dataset.

Limitations of Existing Literature. The above literature review highlights the following limitations, which are *addressed by our methodology for DL-based TC*.

First, it underlines the *scattered nature* of the existing approaches pursuing DL-based TC, as well as their implicit (or partially-justified) design choices. This underlies the lack of a systematic design path, defining the key pillars for the conception and implementation of a practical DL-based mobile TC architecture, and motivates the need for a general DL framework (developed herein) explicitly capitalizing these aspects by molding them into rigorously-defined milestones. Moreover, most of the existing DL-based TC approaches are analyzed in terms of per-class or synthetic classification metrics [21, 91, 92, 94, 101], *without* investigating their performance behavior at a finer-level. Differently, to draw firm conclusions, our performance evaluation also resorts to our systematic evaluation workbench, allowing to compare and assess performance comprehensively, for example from complementary viewpoints (*i.e.* classification performance and complexity) and at different levels of granularity.

On the basis of this comprehensive analysis, we confirm the inability of current DL approaches of consistently outperforming the ML-based ones in realistic and challenging mobile contexts. We trace these deficiencies back to *the use of only a single-modality* in their end-to-end design. As a consequence, in contrast to existing DL-based TC literature, *our MIMETIC framework is designed to exploit different modalities (viz. views or inputs) jointly*, and thus to reap DL promised benefits. Precisely, our proposal is shown to provide a performance improvement in the challenging mobile

scenario with respect to (ML) state-of-the-art [43], single-modal DL baselines [91, 94], and even classifier fusion attempts of their outputs (cf. Chapter 3), thus proving that the gain is due to our prescribed framework, and not to the mere combination of DL algorithms.

5.2 A Framework for Deep Learning-based Mobile Encrypted Traffic Classification

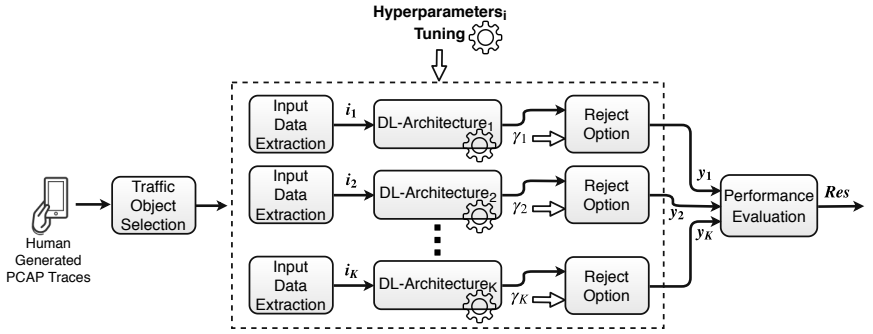


Figure 5.1: Framework for design, tuning, and comparison of DL architectures for TC.

This section analyzes the milestones that should be taken into account for the design of a DL-based (mobile) traffic classifier. We also dissect the application of DL to TC by examining the most common choices made in the state-of-the-art. In detail, the *design milestones* we consider are:

- *Traffic object*: the traffic aggregate atom which induces the segmentation criterion (§5.2.1).
- *Type(s) of input data*: the number and sets of input selected from each traffic object to feed the DL architecture (§5.2.2).
- *DL architecture*: the peculiar DL architecture (*e.g.*, the composition instance of elementary learning layers), coping with input and output

constraints originating from the design choices concerning the *number and type(s) of input data* (§5.2.3).

- *Performance evaluation workbench:* the measures used for a comprehensive performance evaluation of the DL architecture (§5.2.4).

Based on these points, Fig. 5.1 sketches out the framework devised for the systematic design, tuning, and comparison of DL-based traffic classifiers. It is worth pointing out that all the DL classifiers proposed for TC have been carefully analyzed and reproduced. In detail, we have set the hyper-parameter values suggested in their respective works or performed a basic tuning procedure when the latter are not reported. We have leveraged the DL models provided by Keras [170] (Python) API running on top of TensorFlow [171] to implement and test these approaches.

In the next sections, we discuss each design element of our DL-based mobile TC framework separately.

5.2.1 Traffic Object

A key choice regards how raw traffic is segmented into multiple discrete units (cf. §1.3.1). Considering mobile and encrypted traffic, we here suggest the use of either *flows* or *biflows*. Indeed, as reported in Tab. 5.1, most of the related works considered one of these as the relevant objects of classification, with the latter generally achieving better performance with respect to its flow-based counterpart [21, 94]. Other appealing choices are given by the *TCP connection* and the *service burst*. The former differs from the biflow only in the initiation and termination heuristics. The latter has been recently adopted in mobile TC [41, 43] to exploit the bursty traffic-nature and we have also employed it consistently in Chapter 3. Although appealing, a definition of reasonable (and effective) input data for service bursts is not

as straightforward as in the case of (bi)flows given the presence of a varying number of biflows toward the same destination IP/port. Moreover, while there is longstanding practical experience and mature technology working with biflows, using classification results from service bursts becomes hard to translate into actionable and sensible reactions. Still, the service bursts have not seen their direct application to security and policy enforcement so far, as opposed to the ubiquitous (bi)flows. Lastly, in some works (cf. Tab. 5.1) the (finest) object of classification is the *single packet*, entailing the toughest TC task.

5.2.2 Types of Input Data

The recommended types of input data of a generic TC object ingested by DL architectures may be roughly grouped within three categories:

I. The first N_b bytes of payload of traffic object [21, 81, 94].

II. The first N_b bytes of raw data pertaining to the PCAP file related to the traffic object [21, 23, 94].

III. Informative data fields of the first N_p packets [91, 102].

Based on the aforementioned categorization, it is worth noticing that all the types of input data considered for DL are naturally suited for early TC [39].

In the *first* case, the data being fed to the DL architecture is represented by *payload only*, with input data in binary format. In all these works, the payload is arranged in a byte-wise fashion and normalized so as to constrain it within $[0, 1]$. The choice is always justified as a means to reduce the input size for the DL architecture. On the other hand, the layer and size of the payload being chosen depend on the specific work. For example, in [81]

these correspond to the first 1000 bytes of TCP payload. A similar choice is made in [94, 21] for the input labeled as "L7", where 784 bytes from the *application layer* in TCP/IP model are considered. Differently, in [92] and [100] the authors consider the first 1500 and 1480 payload bytes at *layer 2*, respectively, namely the IP header and the first 1480 bytes of each IP payload which results in a 1500 bytes input vector.

The *second* type of input data attempts to gather information from *all protocol layers* (denoted with "ALL" layers in [21, 94]) as in some relevant cases the data from levels lower than layer 7 also contain some useful traffic information (such as transport-layer ports or flags), as pointed out in [21, 94]. Then, since the considered data are typically captured at the *data-link layer*, the payload from frames of *layer* 2 is extracted. However, the traffic provided in this case is always in the form of PCAP files, containing information that could introduce a bias in the classification results.[2] Specifically, in [21, 94] only the first 784 bytes of each TC object are employed. On the other hand, other works chose different input sizes, as 1024 bytes [23] or 900 byte [104][3], justifying the need for careful fine-tuning of the input size (see Sec. 5.4).

Finally, the *third* type of input data is represented by selected protocol fields, not pertaining to the explicit inspection of encrypted payload, of the first N_p packets. For example, in [91], the authors consider only the first 20 packets exchanged into a traffic object (*i.e.* a biflow), and, for each packet, the following 6 fields are extracted (thus a 20×6 matrix is obtained for

[2]We underline that the extraction of "ALL" layers input includes PCAP metadata besides raw packet data (from MAC layer, included). In detail, PCAP global header is of 24 bytes and each packet is also prepended with a 16-byte header, including a timestamp at μs granularity and packet size information.

[3]In this latter case, the authors explicitly remove TCP/UDP headers and MAC addresses with the aim to reduce bias.

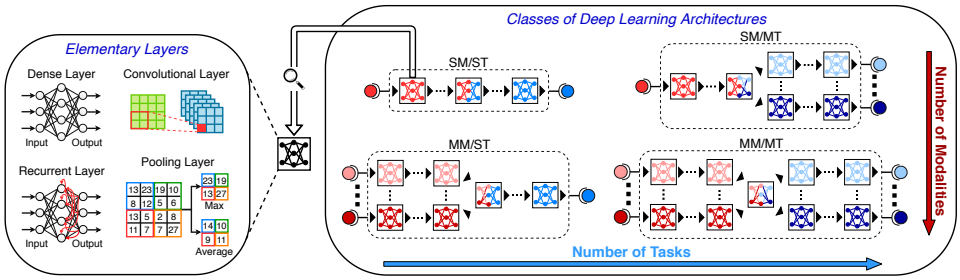


Figure 5.2: Most used elementary layers composing DL architectures and the four different classes of DL architectures, based on single/multiple input modalities (SM/MM) and single/multiple classification tasks (ST/MT).

each traffic object): source and destination ports, number of bytes in transport layer payload, TCP window size[4], IAT, and packet direction ($\in \{0, 1\}$). However, as for the second input type, taking into account port information could lead to biased classification performance. Similarly, in [102], the lengths at the IP layer of the first 128 packets are considered—being however not suited for early TC—whereas in [90] the PL difference, IAT, and packet direction of the first 10 packets are employed. We also highlight that the binary-valued sequence of packets/messages directions has been also recently employed in DL-based WF [25, 47].

Finally, we conclude the discussion mentioning that in all the above cases, there may be instances *longer* or *shorter* than the considered fixed-length (N_b or N_p) data inputs. In such cases, *longer* instances are truncated to the designed length of bytes (N_b) or packets (N_p), in the case of first/second or third type of data, respectively, whereas in the case of *shorter* instances, padding with zeros is always applied in all the discussed works.

5.2.3 Deep Learning-based Classification Architectures

Our framework defines four classes of DL architectures, as shown in Fig. 5.2, based on two orthogonal aspects:

- Whether they are fed with a single type (*single-modal*) or multiple types (*multi-modal*) of input *modalities*, to capitalize complementary viewpoints of the same traffic object (*e.g.*, using the first N_b bytes of transport-level payload together with the informative data fields of the first N_p packets).

- Whether they are in charge of providing inference for one (*single-task*) or multiple (*multi-task*) TC problems (*e.g.*, inferring both the traffic-type and the specific application generating a (bi)flow).

These DL architectures are obtained by composition of *elementary layers* [68], whose common choices are *dense*, *convolutional*, *pooling*, and *recurrent layers* (see Fig. 5.2):

- *Dense layers* (named also *fully-connected*) are the simplest atoms of feed-forward DL architectures, consisting of an affine matrix operation (*i.e.* a linear transformation) on inputs, followed by an entry-wise activation function.

- *Convolutional layers* are the basic building blocks of CNNs, made of a set of translation-invariant filters with a limited extent (*i.e.* the "receptive field") which are convolved with the input, with the aim of extracting the features of a certain input region. Their most common forms adhere to a 1-D or 2-D layout, depending on the specific input nature.

[4]The TCP window size is set to *zero* for UDP packets.

- *Pooling layers* are other key components of CNNs and typically follow a convolutional layer. They perform the down-sampling of the intermediate representations from convolutional layers, with the aim of complexity reduction and overfitting mitigation. Max- and average-pooling are the most common.

- *Recurrent layers* present loopy connections and have in Long Short-Term Memory and Gated Recurrent Unit their most popular variants. These are in charge of recalling values over time, via a state vector, and accept as input a vector sequence. Differently, they output either the final state or its entire time-evolution.

In a nutshell, these elementary layers represent the building blocks of more complex (*i.e. deep*) architectures, that in turn, on the basis of their input/output configuration, belong to one of the above-mentioned classes. Therefore, we now review the architectures obtained from these elementary layers and employed (in their *single-modal* variants) for DL-based (*non mobile*) TC.

For convenience, in the following, we define the m^{th} instance of the training set (made of M samples, with M_ℓ being the number of samples belonging to ℓ^{th} app) as $\boldsymbol{x}_{(m)}$ while the corresponding label with $\ell_{(m)}$, belonging to one among L different classes (*i.e.* $\ell_{(m)} \in \{1, \ldots, L\}$). All the considered DL classifiers are trained to minimize the categorical cross-entropy loss function [68]:

$$\mathcal{L}(\cdot) \triangleq \sum_{m=1}^{M} \left\{ -\sum_{l=1}^{L} t_{l,(m)} \log c_{l,(m)} \right\} \tag{5.1}$$

In the above equation, the one-hot representations of the label $\boldsymbol{c}_{(m)} \triangleq \begin{bmatrix} c_{1,(m)} & \cdots & c_{L,(m)} \end{bmatrix}^T$ and of the corresponding predicted vector $\boldsymbol{t}_{(m)} \triangleq \begin{bmatrix} t_{1,(m)} & \cdots & t_{L,(m)} \end{bmatrix}^T$ are employed. The minimization of the loss $\mathcal{L}(\cdot)$ is

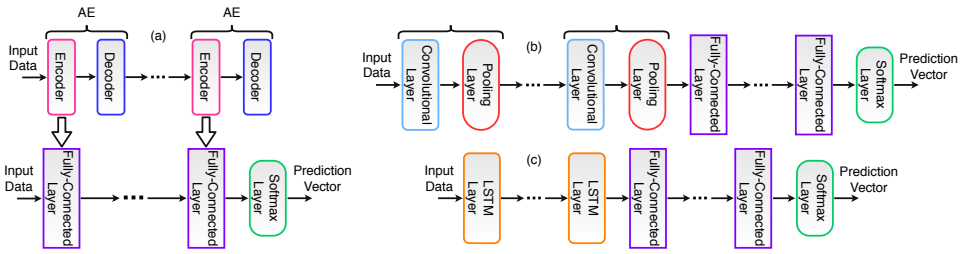


Figure 5.3: DL architectures for TC: SAE (a), CNN (b), and LSTM/GRU (c).

achieved by means of standard (first-order) local optimizers (*e.g.*, stochastic gradient descent, adaptive moment estimation, etc.), resorting to the usual back-propagation for the gradient evaluation (cf. §1.3.4). We also highlight that in a common DL architecture, the last layer is followed by a *softmax* returning a vector that represents the probability distributions of each class.

In Fig. 5.3, we have sketched the following DL-based architectures:

(a) *Stacked AutoEncoder (SAE)*: The SAE (Fig. 5.3a) relies on the basic AE, commonly employed for (unsupervised) feature learning, and whose aim is to (ideally) set the output $\boldsymbol{y}_{(m)} \approx \boldsymbol{x}_{(m)}$, $\forall m = 1, \ldots, M$, by learning a *compressed* data representation. Specifically, the first AE block (*i.e.* the *encoder*) provides a lower-dimensional data representation via a hidden layer of neurons, whereas the second block (*i.e.* the *decoder*) tries to reconstruct the data from the compressed representation. It is worth noticing that the encoder is a particular dense layer.

In practice, to obtain improved performance, a more complex (hierarchical) architecture, namely the SAE, has been proposed [68]. This scheme employs *unsupervised greedy layer-wise pre-training* (top part

of Fig. 5.3a) which stacks up several AEs so that the lower-dimensional representation obtained from j^{th} AE is used as the input of $(j+1)^{th}$ AE (*i.e.* each layer of the network is trained by keeping the weights of lower layers frozen). After training greedily the AE layers, a final softmax layer is added and supervised fine-tuning (*i.e.* a refinement of all layers' weights) of the whole network (bottom part of Fig. 5.3a) for the classification task is performed (*i.e.* using $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(M)}$ along with $\ell_{(1)}, \ldots, \ell_{(M)}$). A relevant application of SAE to TC is found in [92], consisting of *five* stacked layers—with $\{400, 300, 200, 100, 50\}$ neurons and 25% dropout-probability [68] after each layer (to mitigate over-fitting)—all employing rectified linear unit activations.

(b) *Convolutional Neural Networks (CNNs)*: The CNNs (Fig. 5.3b) are widely-used DL models, inspired by visual mechanism of living organisms, and made of chained convolutional and (typically) pooling layers. The higher layers of a CNN are usually a few dense layers similar to those of the AE compressing stage, with the last having the essential softmax activation. For example, the architecture in [94] is made of two 1D convolutional layers (with 32 and 64 filters, respectively), each followed by a 1D max-pooling, and terminated with two dense layers.

Similarly, the CNN in [21] is obtained by replacing 1D with 2D (pooling/convolutional) layers and interpreting the input as a "traffic image". A similar 2D-CNN is also considered in [91], where *batch normalization* [68] is also applied after each max-pooling layer. Differently, in [92] a 1D-CNN consisting of two 1D convolutional layers (200 and 80 filters, respectively, with 1D average-pooling) and seven fully-connected layers (with $\{600, 500, 400, 300, 200, 100, 50\}$ neurons), all

having rectified linear unit activations, is considered. Additionally, to avoid the over-fitting, 25% dropout after the pooling layers and *early stopping* technique are adopted [68].

(c) *Long Short-Term Memory (LSTM) / Gated Recurrent Unit (GRU)*: An LSTM (Fig. 5.3c) is a popular (easier to train) variant of recurrent neural networks having unit connections forming a directed cycle and being able to model *dynamic* temporal behaviors with "long-term dependencies" [68]. A neural network made of LSTM units is often called an LSTM network.

An LSTM unit is in charge of "remembering" values (via a state vector $\boldsymbol{h}[t]$) over arbitrary time intervals and is composed of a *cell*, *input*, *output*, and *forget* gates, while having as input a vector sequence of length T: $\boldsymbol{x}[1], \ldots, \boldsymbol{x}[T]$ (*i.e.* each training instance is a matrix). The final hidden state $\boldsymbol{h}[T]$ corresponds to the output of the LSTM unit. A standard LSTM network for classification is usually terminated with a few dense layers, with the last having a softmax activation. On the other hand, when several LSTM layers are stacked, they expose as output (except for the last one) the finer-grained time-evolution of the state vs. the input sequence, $\boldsymbol{h}[1], \ldots, \boldsymbol{h}[T]$ (modeling a "return-sequences" behavior), forming the input to the higher LSTM layer.[5]

Similarly, a GRU unit is in charge of modeling long-term dependencies like the LSTM, but it operates using only *reset* and *update* gates. Consequently, a GRU network has less parameters to train [68] and thus it is computationally more efficient (*i.e.* it uses less memory and it can

[5]We highlight that for successive LSTM layers, the temporal-dimension of data-input does not change, whereas the vector-size of the successive inputs does, being function of the size of the hidden state.

be trained faster). It should be noted that LSTM and GRU units can be also conceived in an improved "bidirectional" form, namely their internal representation is split into forward and backward directions.

For example in [91] a standard LSTM ending with two fully-connected layers of 100 and 108 nodes (the latter being the number of services to discriminate from) is considered. Interestingly, a stack of LSTM layers is also proposed in [91] in the context of hybrid architectures, as described henceforth.

Indeed, the discussed elementary learning layers can be also jointly employed within a single *hybrid DL architecture*. For example, architectures based on the combination of 2D convolutional and LSTM layers may be conceived [91], where the output tensor of the convolutional layer is reshaped into a matrix fed as input to an LSTM unit.

5.2.4 Performance Evaluation Workbench

The proposed comparison framework includes the following common performance measures (see §1.3.2): (*i*) accuracy, (*ii*) precision, (*iii*) recall, and (*iv*) specificity. Since the latter three are defined on a per-app basis, we consider the *F-measure* and the *G-mean* so as to account for their effects concisely, and employ their arithmetically averaged (viz. macro) versions. We consider also the *Top-K accuracy* to analyze the soft-output of a DL classifier and report the *confusion matrices* with the aim of recognizing the most frequent misclassification patterns. Additionally, we test the classifiers when equipped with a reject option (with threshold γ) to evaluate the effectiveness of tuning γ for improving classification performance with respect to decreasing in CR.

Moreover, to further deepen soft-output behavior, this performance evaluation workbench enables a *calibration analysis*, that allows to check whether the class-probability estimates are representative of the true-class (posterior) probabilities. Indeed, a miscalibrated classifier produces confidences (*i.e.* class-prediction probabilities) that could not represent the true probabilities, leading to either excessively optimistic or pessimistic decisions. Specifically, we leverage *reliability diagrams* that show the accuracy as a function of confidence and are obtained by partitioning the predictions into M equally-spaced bins and calculating the accuracy of each bin. If the classifier is perfectly calibrated, then the diagram should plot the identity function (*e.g.*, operating with 70% confidence leads to 70% accuracy) and any deviation from a perfect diagonal represents a miscalibration. In addition to reliability diagrams, for conciseness we report also the *Expected Calibration Error (ECE)*. The latter measure is defined as the weighted (based on the number of samples) mean, evaluated over all the bins, of the difference between accuracy and confidence [172].

For completeness, as a preliminary investigation of the computational complexity of DL-architectures' training phase, we report their training time, given the specificity of such phase in mobile TC, due to apps' fingerprint aging because of their (and OS) updates. Precisely, since training is performed on multiple epochs [68], we report such info in a terse (normalized) way, by providing the Run-Time Per-Epoch (RTPE).

Finally, for each considered analysis, our evaluation is based on a (stratified) ten-fold cross-validation, (*i*) representing a stable evaluation process and (*ii*) maintaining the same share of class imbalance in both training and test sets within each fold. Accordingly, we report both the mean and the variance of each performance measure as a result of the evaluation on the ten different folds.

5.3 The MIMETIC Architecture

In the previous section, we have sketched the design milestones for the fruitful application of DL to (mobile) TC, dissecting the most common choices made in related literature according to these milestones, and defining a framework for tuning and comparison of DL-based traffic classifiers. In view of these considerations, herein, we resort to the proposed framework to capitalize the multi-modal nature of traffic data by means of a novel *multi-modal DL-based mobile traffic classification (MIMETIC)* architecture.

We introduce MIMETIC, starting from the high-level architectural description, in Sec. 5.3.1. We then focus, in Sec. 5.3.2, on the general procedure adopted for training it. Both the architectural description and the training procedure are shown in Fig. 5.4 from a conceptual standpoint. Finally, in Sec. 5.3.3, we focus on the specific instance of MIMETIC (see Fig. 5.5) chosen and evaluated in later Sec. 5.4.

In the following, we refer to the same notation introduced in Sec. 5.2.3. Overall, Tab. 5.2 describes the mathematical notations used to define the MIMETIC architecture.

5.3.1 Architectural Overview

In Sec. 1.3.4, we have have shown that DL approaches—as opposed to ML-based ones—are able to learn app fingerprints in an *end-to-end* fashion, that is directly from the type of input selected, thus defeating the tedious and lowly-adaptable process of feature design. However, the traffic data is highly-structured by definition, as it contains information referring to the whole protocol stack. As a result, a monolithic DL architecture taking the whole information coming from a TC object in bulk—*early* (or *data*) *fusion*—is likely to be suboptimal, since the parameter set would overfit to

Table 5.2: List of the mathematical notations used to define the MIMETIC architecture.

Symbol	**Definition**
M	Number of training samples
M_ℓ	Number of training samples of the ℓ^{th} app
P	Number of different inputs (modalities)
J_p	Number of single-modality layers
$\boldsymbol{x}_{(m)}$	m^{th} sample of the training set
$\ell_{(m)}$	Label (true class) of $\boldsymbol{x}_{(m)}$
$\boldsymbol{t}_{(m)}$	One-hot representation of $\ell_{(m)}$
$\boldsymbol{c}_{(m)}$	Predicted class confidences of $\boldsymbol{x}_{(m)}$
$\mathrm{CE}(\boldsymbol{t}, \boldsymbol{c})$	Categorical cross-entropy between $\boldsymbol{t}$ and $\boldsymbol{c}$
w_m	Weight assigned to $\boldsymbol{x}_{(m)}$
$\boldsymbol{\theta}_p$	Parameters of the p^{th} single-modality layers
$\boldsymbol{\theta}_p^{\uparrow}$	Parameters optimized in pre-training and fine-tuning
$\boldsymbol{\theta}_p^{\downarrow}$	Parameters optimized only in pre-training
$\boldsymbol{\theta}_p^{\mathrm{stub}}$	Parameters of the p^{th} "stub" layer
$\boldsymbol{\theta}_0$	Parameters of the shared representation layers
$\hat{\boldsymbol{\theta}}_p$	Pre-trained parameters of the p^{th} single-modality layers
$\hat{\boldsymbol{\theta}}_p^{\mathrm{stub}}$	Trained parameters of the p^{th} "stub" layer
$\mathcal{L}_p(\cdot)$	Loss function minimized in pre-training of p^{th} modality
$\mathcal{L}(\cdot)$	Loss function minimized in fine-tuning
$\boldsymbol{h}[t]$	State vector of recurrent layers

one input subset while underfitting the others. Differently, the capitalization of score-results—*late* (or *score/decision*) fusion or *multi-classification*—of DL-based traffic classifiers built on different modalities, although effective in some cases (cf. Chapter 3), is not able to fully exploit the benefits of multi-modality (as also shown experimentally in Sec. 5.4). Based on these reasons, multi-modal DL is here foreseen as an appealing alternative toward a sophisticated form of information fusion, named *intermediate fusion* [166], overcoming both the limitations of the *early* (or *data*) *fusion* and *late* (or *score/decision*) *fusion*, offering a truly-flexible tool for practical mobile TC

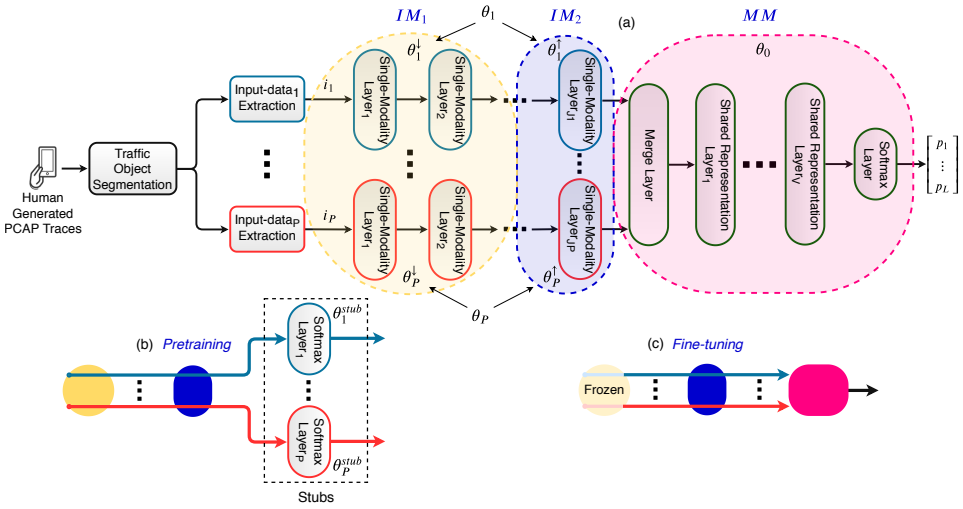


Figure 5.4: General illustration of the MIMETIC architecture. (a) depicts the architecture by highlighting single-modality representation layers, differentiated as those that are only pre-trained (IM_1) and those that are also fine-tuned (IM_2), and shared representation-layers (MM), along with the corresponding parameter set. (b) and (c) depict the proposed training procedure based on pre-training and fine-tuning.

enjoying multi-modality. The description of the MIMETIC architecture is provided hereinafter.

As sketched in Fig. 5.4a, at an abstract level, the architecture of MIMETIC is fed with P different inputs (*modalities* or *views*) for each traffic object to be classified, with the p^{th} modality provided from $Input\text{-}data_p$ extraction block. Such deep network architecture is firstly composed of J_p *single-modality* (input-specific) *layers* (depicted with different colors, *i.e.* blue and red, in Fig. 5.4), allowing to extract in an increasingly-abstract fashion the discriminative features pertaining to the p^{th} view, capitalizing intra-modality dependence. Specifically, the set of parameters referring to single-modality layers of the p^{th} modality is referred to as $\boldsymbol{\theta}_p$. On top of

these layers, the abstract features are joined via a *merge layer* (in green in Fig. 5.4), which represents the first layer channeling the modality-specific distilled information toward a joint multi-modal representation. Although the most general (and common) choice is represented by a *concatenation operation*, other approaches may be pursued in case the abstract features originating from different modalities have the *same size*, for example averaging, entry-wise maximum, etc.

Finally, the architecture is completed with a few *shared representation layers* (also in green), distilling the features capturing inter-modality dependencies, and the usual *softmax layer*, returning the soft-output vector for the mobile TC task considered. Hereinafter, the set of parameters related to shared representation layers (plus the final softmax) is referred to as $\boldsymbol{\theta}_0$. Also, to promote *regularization* (so as to avoid overfitting), dropout between successive layers and early-stopping techniques are adopted [68]. Single-modality and shared-representation layers are commonly implemented choosing from the elementary layers prescribed in our framework and described in Sec. 5.2.3.

Given the general definition of the MIMETIC architecture, we next describe the algorithmic procedure for its training.

5.3.2 Proposed Training Procedure

The high-level procedure suggested for training a classifier in the MIMETIC architecture is shown as pseudocode in Algorithm 1 and described hereafter.

The architecture of MIMETIC is trained via a *two-stage phase*, made of *pre-training* and *fine-tuning* [68]. The reason for a preliminary pre-training procedure is to correctly distill discriminative information from each modality so as to capitalize the advantage of the multi-modal traffic representation. Before proceeding, we recall that training of DL ap-

Algorithm 1 Pseudo-code of MIMETIC Training Procedure.

```
/* pre-training */
1 for Modality p ∈ [1, P] do      /* parallelizable */
2 |  ( θ̂_p, θ̂_p^stub ) ← trainSingleM(θ_p, θ_p^stub, TrainingSet)
  |  /* θ̂_p^stub is discarded */
3 |  [θ̂_p^↓ θ̂_p^↑] ← θ̂_p ;                          /* θ̂_p is split */
4 end
/* fine-tuning */
5 θ^↓ ← θ̂^↓_{1,...,P} ;                                  /* frozen */
6 θ^↑ ← θ̂^↑_{1,...,P} ;                              /* initialized */
7 ( θ_0, θ^↑ ) ← trainMultiM(θ_0, θ^↓, θ^↑, TrainingSet)
```

proaches resort to the "one-hot" representation [68] of each label $\ell_{(m)}$, namely $\boldsymbol{t}_{(m)} \triangleq [t_{1,(m)}, \cdots, t_{L,(m)}]$, whose entries are all zero, save from a single "1" corresponding to $\ell^{th}_{(m)}$ class.

Specifically, each single-modality stack is first (pre-)trained independently, that is *without* the shared representation layers and by topping each modality chain with a softmax layer "stub", whose parameters are collected within $\boldsymbol{\theta}_p^{\text{stub}}$—see Alg. 1 lines 1–4 and Fig. 5.4b. Specifically, the p^{th} "stubbed" chain is trained to minimize the classification loss function $\mathcal{L}_p(\cdot)$ with the intent of promoting p^{th} modality capability to solve the TC task *alone*, defined as:

$$\mathcal{L}_p\left(\boldsymbol{\theta}_p, \boldsymbol{\theta}_p^{\text{stub}}\right) = \sum_{m=1}^{M} w_m \, \text{CE}(\boldsymbol{t}_{(m)}, \, \boldsymbol{c}_{(m)}[\boldsymbol{\theta}_p, \boldsymbol{\theta}_p^{\text{stub}}]) \tag{5.2}$$

We recall that the vector $\boldsymbol{c}_{(m)} \triangleq [c_{1,(m)}, \cdots, c_{L,(m)}]$ collects the predicted class confidences of DL classifier (which depend on the network parameters) for the label of the m^{th} training sample. These confidences should be as close as possible to the (ground-truth originated) one-hot vector

$\boldsymbol{t}_{(m)} \triangleq [t_{1,(m)}, \cdots, t_{L,(m)}]$. Such distance is measured via $\text{CE}(\boldsymbol{t}, \boldsymbol{c}) \triangleq -\{\sum_{\ell=1} t_\ell \log c_\ell\}$, denoting the *categorical cross-entropy* of the m^{th} training sample. Furthermore, differently than the categorical cross-entropy commonly minimized by (single-modal) DL classifiers (as in defined in Sec. 5.2.3), MIMETIC includes the minimization of a general *weighted* form of the categorical cross-entropy, with w_m denoting the weight of the m^{th} sample, enabling *cost-sensitive learning* [68]. Indeed, the weight w_m allows penalizing/favoring, during training phase, the discrimination capability toward some app(s) and/or mitigating the class-imbalance problem. The learned parameters from the above optimization are indicated with $(\hat{\boldsymbol{\theta}}_p, \hat{\boldsymbol{\theta}}_p^{\text{stub}})$. Specifically, for each p^{th} modality, the learned parameter set $\hat{\boldsymbol{\theta}}_p$ is split in $[\hat{\boldsymbol{\theta}}_p^{\downarrow}\ \hat{\boldsymbol{\theta}}_p^{\uparrow}]$, corresponding to pre-trained parameters of low-layers (IM_1 in Fig. 5.4a) and high-layers (IM_2 in Fig. 5.4a) in DL hierarchy, respectively.

Then, during the *fine-tuning* phase—see Fig. 5.4c and Alg. 1 lines 5–7—the above softmax stubs are removed (*i.e.* $\hat{\boldsymbol{\theta}}_1^{\text{stub}}, \cdots, \hat{\boldsymbol{\theta}}_P^{\text{stub}}$ are discarded from the optimization before actual fine-tuning) and the training of the *whole* MIMETIC architecture is performed (*i.e.* including both the parameters of single-modality layers $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_P$ and of shared-representation layers $\boldsymbol{\theta}_0$, associated to the MM block). However, as a result of the pre-training phase, a share of single-modality layers (*i.e.* those corresponding to low-layers in DL hierarchy, named IM_1) are typically *frozen* when fine-tuning phase is performed. This is due to the fact that the low-level layers refer to *intra-modality automatic* feature extraction [165]. In other terms, within $\boldsymbol{\theta}_P \triangleq \left[\boldsymbol{\theta}_p^{\downarrow} \quad \boldsymbol{\theta}_p^{\uparrow}\right]$ only the subset $\boldsymbol{\theta}_p^{\uparrow}$ is (further) optimized during fine-tuning (*i.e.* those corresponding to IM_2), while $\boldsymbol{\theta}_p^{\downarrow}$ is kept *fixed* to the value learned during pre-training, that is $\boldsymbol{\theta}_p^{\downarrow} = \hat{\boldsymbol{\theta}}_p^{\downarrow}$. As a result, the following *weighted* form

of the *categorical cross-entropy* loss function is minimized:

$$\mathcal{L}\left(\boldsymbol{\theta}_1^{\uparrow}, \cdots, \boldsymbol{\theta}_P^{\uparrow}, \boldsymbol{\theta}_0\right) \triangleq \sum_{m=1}^{M} w_m \, \mathrm{CE}(\boldsymbol{t}_{(m)}, \boldsymbol{c}_{(m)}[\boldsymbol{\theta}_1^{\uparrow}, \cdots, \boldsymbol{\theta}_P^{\uparrow}, \boldsymbol{\theta}_0]) \tag{5.3}$$

The loss functions concerning pre-training and fine-tuning phases (*i.e.* $\mathcal{L}_p(\cdot)$ and $\mathcal{L}(\cdot)$, respectively) are minimized via standard first-order local optimizers (*e.g.*, SGD, ADAM, etc.), resorting to the usual back-propagation for gradient evaluation [68].

We now present the specific instance obtained from MIMETIC framework and used for the experimental evaluation.

5.3.3 Implementation of a Traffic Classifier based on MIMETIC

The specific implementation[6] of the proposed MIMETIC architecture (see Fig. 5.5) operates at *biflow level*—as prescribed in our proposed framework and aiming at a consistent comparison with most of earlier works (see Tab. 5.1) employing single-modal DL for TC—and is made of $P = 2$ *modalities*. Based on the analysis carried out in Sec. 5.2.2, these are fed with the corresponding two types of input prescribed in our framework, that are naturally suited for early TC and have been already employed successfully in most related works performing TC via single-modal DL: (I) the first N_b bytes (normalized within $[0, 1]$) of payload; (III) informative protocol fields—namely: number of bytes in transport-layer payload, TCP window size (set to zero for UDP), IAT, and packet direction ($\in \{0, 1\}$)—of the first N_p packets. We remark that we focus on payload at the application-

[6]We highlight that consistently with the DL models employed for single-modal DL classifiers described in Sec. 5.2.3, we have leveraged *Keras* [170] (Python) API running on top of *TensorFlow* [171] to implement and test the MIMETIC instance described in this section.

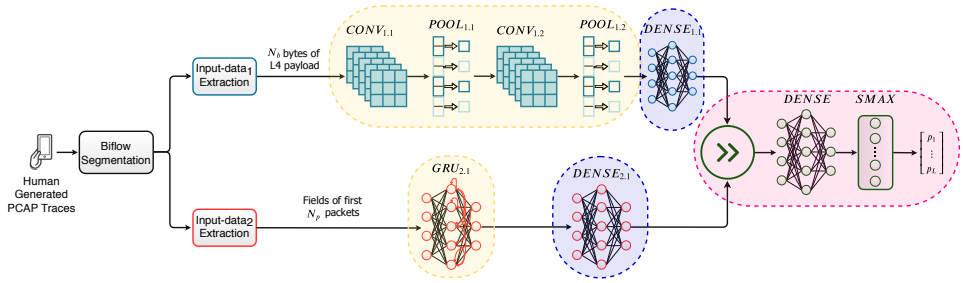


Figure 5.5: Implementation of considered traffic classifier based on the MIMETIC architecture. Background colors specify the groups of layers IM_1 (in yellow), IM_2 (in blue), and MM (in purple).

layer (viz. L4 payload) in TCP/IP stack in (I) and, also, we do not consider (II) input type (viz. the first N_b bytes of raw data belonging to the PCAP file) as well as port info in (III), as otherwise these latter may both lead to biased and inflated performance, as we show in Sec. 5.4.1. Finally, pre-processing operations prescribed in Sec. 5.2.2 are also performed on instances longer/shorter than the designed length of bytes (N_b) or packets (N_p). We underline that the above "traffic-originated" modalities refer to different levels of abstraction (packet vs. biflow depth) and standpoints (encryption-dependent vs. encryption independent) for the observed traffic. This inspired the adoption of a multi-modal architecture to improve classification performance, as later supported by the experimental validation in Sec. 5.4. For the mentioned reasons, we parallel the use of both modalities as for audio and video modalities in natural language understanding.

Hereinafter, we refer to the building blocks of Fig. 5.5 to describe the specific implementation of the proposed MIMETIC architecture. The single-modality layers of the *first view* (the "payload" modality) are two 1D convolutional layers ($CONV_{1.1}$ and $CONV_{1.2}$, made of 16 and 32 filters, respectively, with kernel size of 25, unit stride, and rectified linear unit activa-

tions), each followed by a 1D max-pooling layer ($POOL_{1.1}$ and $POOL_{1.2}$, with unit stride and spatial extent equal to 3) and, finally, by one dense layer ($DENSE_{1.1}$, with 256 nodes). The reason for this choice is the ability of 1D convolutional layers to extract spatially-invariant (discriminative) patterns from the payload. On the other hand, the single-modality layers of the *second view* (the "protocol fields" modality) are, in order, a bidirectional GRU ($GRU_{2.1}$, with 64 nodes and return-sequences behavior) and one dense layer ($DENSE_{2.1}$, with 256 nodes). Such choice was driven by the GRU ability to capture long-term dependencies pertaining to the initial segments of the biflow, while requiring slightly-less parameters with respect to the more common LSTM (cf. §5.2.3). The intermediate features of the two branches are then concatenated via a *merge layer* ($>>$), and fed to a *dense (shared-representation) layer* ($DENSE$, with 128 nodes), before the softmax ($SMAX$). In all the layers, the outputs are obtained via rectified linear unit activations. Finally, 20% dropout is applied after (a) each dense layer (including the merge layer) and (b) after flattening the 2D representation of both the stack of convolutional/pooling layers and GRU.

The considered architectural instance is *trained* via the *two-stage phase* described in Sec. 5.3.2. Specifically, all the classification loss functions ($\mathcal{L}_1(\cdot), \cdots, \mathcal{L}_P(\cdot)$ and $\mathcal{L}(\cdot)$) include cost-sensitive learning, here exploited to mitigate natural class imbalance found in mobile traffic (cf. §2.5). To this end, the weight $w_m \triangleq M/M_{\ell(m)}$ is assigned to m^{th} sample, being inversely proportional to the number of training-set samples labeled with $\ell_{(m)}$, thus magnifying (resp. decreasing) the contribution of apps with a few (resp. high) number of samples. Concerning the *pre-training* phase, each single-modality stack is first (pre-)trained independently for 25 epochs each by topping a softmax layer stub and by minimizing the loss $\mathcal{L}_p(\cdot)$ (cf. Eq. (5.2)), so that mobile TC could be performed on either (transport-layer)

payload or protocol fields. Then, *fine-tuning* of the whole multi-modal DL architecture is performed (for 40 epochs) after freezing IM_1 (*i.e.* the low-layers in DL hierarchy), namely the convolutional and recurrent layers, $CONV_{1.1}/CONV_{1.2}$ and $GRU_{2.1}$, respectively, and by minimizing the loss $\mathcal{L}(\cdot)$ (cf. Eq. (5.3)). For both phases, we have employed the ADAM optimizer (batch size of 50) and the early-stopping technique (to prevent overfitting) measured on the training accuracy. We underline that the overall number of epochs ($25 \times 2 + 40 = 90$) has been chosen by considering the values suggested in more-related works [21, 91, 94] so as to keep the complexity low, while properly training the architecture.

5.4 Experimental Evaluation

This section investigates and compares the performance (from both classification and complexity standpoints) of existing single-modal DL classifiers (§5.4.1) leveraging the framework—encompassing the evaluation workbench—proposed in Sec. 5.2. This analysis enables to both outline key guidelines for the design of effective and unbiased DL-based mobile and encrypted traffic classifiers and determine the best-performing ones to be used as (one of the) baselines for the assessment of the performance of devised MIMETIC approach (§5.4.2). The evaluation is consistently based on the common benchmark consisting of the three mobile traffic datasets described in Sec. 2.5.1.

5.4.1 Single-modal Deep Learning-based Classifiers

As just mentioned, we begin our investigation with the systematic comparison of the considered single-modal DL architectures. As indicated in Sec. 5.2, the architectures are trained for 90 epochs, following the suggestions in related studies [21, 47, 81, 91, 94], with the ADAM optimizer (batch

size of 50) and early-stopping technique measured on the training accuracy. For completeness, *two baseline approaches* are also included in our analysis of classification efficacy:

- The flow-based RF developed in [42, 43], that we have already employed and tested in Chapter 3 (referred to as *Tay_RF*), showing that it outperforms other standard ML-based mobile-traffic classifiers when fed with 40 carefully-handcrafted statistical features[7]. Therefore, such flow-based RF represents the current state-of-the-art mobile-traffic classifier, but is applicable only in the case of "post-mortem" TC, as opposed to inputs used in DL classifiers, suited for early TC.

- A MLP with only one hidden layer (with 100 nodes), here denoted as MLP-1, corresponding to a lower-bound on achievable performance and trained on the same inputs as single-modal DL architectures, so as to stress the performance achievable by "shallow" learning in the same setup. Aiming at a consistent comparison, we have used the same number of epochs (*i.e.* 90), optimizer (*i.e.* ADAM), and batch size (*i.e.* 50) for training of MLP-1 (along with early-stopping).

Hereinafter, for compactness we refer to type (I) (resp. type (II)) input data corresponding to the first N_b bytes of payload (resp. raw) data as "L7-N_b" (resp. "ALL-N_b") [21, 81, 94]. Differently, the $N_p \times 4$—indicated as $N_p \times 6$ and highlighted through a "$\star$" marker, when ports are included—input matrix of protocol fields extracted from each biflow (*i.e.* type (III)) is denoted with "MAT-N_p" [91]. For example, considered single-modal

[7]We recall that these correspond to the 40 best-ranked statistics (*i.e.* min, max, mean, standard deviation, variance, mean absolute deviation, skewness, kurtosis, and percentiles) based on the Gini impurity score and computed on the sets of upstream, downstream, and complete (*i.e.* both of them) IP packet lengths (see [42] and §3.2.1).

DL classifiers adopted $N_b = 1000$ [81] or $N_b = 784$ [21, 94] bytes and $N_p = 20$ [91] packets, respectively. Finally, for consistency, the first N_p packet directions (*i.e.* type (III)) are reported as "DIR-N_p" [47]. We refer to Sec. 5.2.2 for further details on input data employed.

Table 5.3: Accuracy, F-measure, and G-mean [%] of single-modal DL-based and baseline traffic classifiers. Results refer to the FB/FBM dataset and are in the format *avg. (± std.)* obtained over 10-folds. Results with diamonds (◇) and stars (⋆) refer to *biased* inputs and inputs *including TCP/UDP ports*, respectively. **Best-performing** DL-based and shallow classifiers fed with *unbiased* inputs are highlighted.

Architecture	*Accuracy*	*F-measure*	*G-mean*
SAE [92] (L7-1000)	73.52 (± 0.82)	71.82 (± 1.31)	70.49 (± 2.25)
2D-CNN [21] (L7-784)	75.56 (± 3.15)	73.95 (± 2.54)	71.81 (± 2.07)
2D-CNN [21] (ALL-784) ◇	73.99 (± 3.03)	72.54 (± 2.80)	70.85 (± 3.33)
1D-CNN [94] (L7-784)	**76.37 (± 0.73)**	**75.56 (± 1.01)**	**74.79 (± 1.76)**
1D-CNN [94] (ALL-784) ◇	75.91 (± 2.74)	75.53 (± 2.68)	75.46 (± 2.61)
2D-CNN [91] (MAT-20) ⋆	71.82 (± 1.13)	70.84 (± 1.12)	70.01 (± 1.07)
LSTM [91] (MAT-20) ⋆	72.59 (± 0.75)	71.76 (± 0.78)	71.10 (± 0.85)
HYBRID [91] (MAT-20) ⋆	72.36 (± 0.95)	71.41 (± 0.96)	70.58 (± 1.04)
2D-CNN [91] (MAT-20)	73.33 (± 0.93)	72.18 (± 1.04)	71.02 (± 1.16)
LSTM [91] (MAT-20)	73.54 (± 0.49)	72.50 (± 0.58)	71.49 (± 0.85)
HYBRID [91] (MAT-20)	74.26 (± 0.98)	73.23 (± 0.95)	72.18 (± 1.05)
2D-CNN [47] (DIR-784)	66.51 (± 0.57)	63.88 (± 0.82)	61.28 (± 1.23)
MLP-2 [47] (DIR-784)	58.93 (± 0.80)	56.65 (± 2.20)	54.73 (± 3.83)
MLP-1 (L7-1000)	73.78 (± 1.09)	72.58 (± 1.16)	71.95 (± 1.43)
MLP-1 (L7-784)	**74.46 (± 0.88)**	**73.89 (± 0.86)**	**73.55 (± 0.89)**
MLP-1 (ALL-784) ◇	76.39 (± 0.96)	75.82 (± 0.90)	75.42 (± 0.91)
MLP-1 (MAT-20) ⋆	68.66 (± 0.99)	67.65 (± 1.13)	66.88 (± 1.45)
MLP-1 (MAT-20)	68.93 (± 1.32)	67.86 (± 0.94)	66.98 (± 0.75)
Tay_RF [42] (biflow-based)	79.56 (± 0.62)	78.73 (± 0.62)	78.37 (± 0.76)

"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

Biased vs. Unbiased Input Types. First, in Tabs. 5.3 and 5.4 we report the results of state-of-the-art single-modal DL-based (and baseline)

approaches fed with inputs (and features) extracted from binary FM/FBM dataset and multi-class Android and iOS datasets, respectively. We highlight that the performance of classifiers marked with diamond ($\diamond$) and star ($\star$) markers represents results from *biased inputs* and inputs *including source and destination ports* (cf. §5.2.2) and, therefore, they *should not* be considered as *meaningful elements of comparison*. Indeed, a DL classifier fed with all the data contained in a packet or in a set of packets (*e.g.*, "ALL-N_b" input), and thus overlooking the presence of PCAP metadata, likely leads to *misleading performance results*. Similarly, the input including port numbers yields DL statistical port-based architectures. Furthermore, whether destination port may be useful in some "static" contexts, this is never the case for the source port, which is subject to a choice depending on sequential numbering or, in a more sophisticated fashion, to randomization.

From the inspection of results it is apparent that, referring to the FB/FBM dataset (cf. Tab. 5.3), only the 1D-CNN (L7-784) is able to outperform the shallow classifiers MLP-1 (L7-1000/L7-784) in terms of all the metrics analyzed (*i.e.* +1.91%, +1.67%, and +1.24% in terms of accuracy, F-measure, and G-mean, respectively). This result confirms the intuition that discriminative information from *traffic should be extracted by naturally considering data as one-dimensional* (viz. time-series) as compared to those feeding 2D-CNN (L7-784). Nonetheless, in the binary dataset neither the best DL classifier is able to achieve performance comparable with the flow-based Tay_RF. This may be attributed to the need of a more informative type of input or more complex approaches (*e.g.*, MIMETIC), providing a higher discriminative power in the case of very similar apps, like FB and FBM. Differently, focusing on the DL approaches with "MAT-20 ($\star$)" input, the results show that FB/FBM classification task is almost *port-independent*, exhibiting even a slight performance gain—*e.g.*, +1.91%

accuracy with the HYBRID (viz. LSTM + 2D-CNN) (MAT-20) approach—when ports are removed. This may be the consequence of high port randomization or/and (likely) use of overlapping port sets (*e.g.*, corresponding to common Facebook-platform services).

On the other hand, referring to multi-class datasets (cf. Tab. 5.4), DL approaches are able to provide improved performance with respect to shallow classifiers with analogous unbiased inputs, namely MLP-1 (L7-1000/L7-784/MAT-20), and even outperform flow-based state-of-the-art RF. This is ascribed to DL ability to implicitly learn very complex features able to distinguish (seemingly) similar traffic generated from different apps and thus motivating the strong appeal of DL in this challenging scenario. Indeed, in the Android setup, 85.70% accuracy, 78.68% F-measure, and 86.82% G-mean are achieved by 1D-CNN (L7-784), as opposed to 84.78%, 75.49%, and 83.86%, respectively, obtained by Tay_RF. We also notice that, in both datasets, 1D-CNN (L7-784) achieves very similar performance to 2D-CNN (L7-784), still proving the benefit in considering one-dimensional traffic data.

A similar reasoning applies to the iOS case, although the LSTM performs the best in terms of the three considered metrics, but only when *port information is taken into account* (*i.e.* with "MAT-20 $\star$" input). Differently, a significant performance drop is observed for each DL classifier with "MAT-20" input compared to its counterpart including both source and destination TCP/UDP ports in the input ("$\star$" marker), highlighting a different trend with respect to the FB/FBM dataset. For example, up to -19.68% in F-measure is observed for multi-class datasets, with the worst drop affecting precisely the LSTM in the iOS setup, confirming the assumption that including port-information could lead to deceptive performance rise. Finally, the directions of packets belonging to a biflow (albeit representing

Table 5.4: Accuracy, F-measure, and G-mean [%] of single-modal DL-based and baseline traffic classifiers. Results refer to the the multi-class datasets and are in the format *avg. (± std.)* obtained over 10-folds. Results with diamond (◇) and star (⋆) markers refer to *biased* inputs and inputs *including TCP/UDP ports*, respectively. **Best-performing** DL-based and shallow classifiers fed with *unbiased* inputs are highlighted for both datasets.

Architecture	*Android*			*iOS*		
	Accuracy	*F-measure*	*G-mean*	*Accuracy*	*F-measure*	*G-mean*
SAE [92] (L7-1000)	75.15 (± 1.52)	57.00 (± 2.78)	69.07 (± 3.42)	74.55 (± 0.80)	60.57 (± 2.06)	74.86 (± 1.89)
2D-CNN [21] (L7-784)	85.46 (± 0.48)	**78.78 (± 1.39)**	**86.92 (± 1.26)**	**82.72 (± 1.47)**	**74.41 (± 0.90)**	83.91 (± 0.95)
2D-CNN [21] (ALL-784) ◇	95.74 (± 0.24)	92.05 (± 0.65)	95.15 (± 0.56)	95.27 (± 1.19)	92.48 (± 0.91)	95.41 (± 0.76)
1D-CNN [94] (L7-784)	**85.70 (± 0.45)**	78.68 (± 1.20)	86.82 (± 0.87)	82.64 (± 1.63)	74.34 (± 1.29)	**84.00 (± 1.31)**
1D-CNN [94] (ALL-784) ◇	95.73 (± 0.67)	92.18 (± 1.19)	95.42 (± 1.02)	95.97 (± 0.38)	92.33 (± 0.99)	95.45 (± 0.67)
2D-CNN [91] (MAT-20) ⋆	82.22 (± 0.42)	70.81 (± 0.97)	82.18 (± 0.79)	81.23 (± 0.73)	73.04 (± 1.33)	83.64 (± 1.03)
LSTM [91] (MAT-20) ⋆	81.18 (± 0.41)	69.68 (± 0.81)	81.21 (± 0.65)	83.54 (± 0.64)	75.95 (± 1.11)	85.88 (± 0.89)
HYBRID [91] (MAT-20) ⋆	83.53 (± 0.41)	72.02 (± 0.77)	82.51 (± 1.01)	82.28 (± 0.42)	74.22 (± 0.93)	84.36 (± 0.92)
2D-CNN [91] (MAT-20)	76.01 (± 0.70)	62.83 (± 1.28)	75.60 (± 1.29)	68.53 (± 0.61)	58.67 (± 1.22)	72.95 (± 1.30)
LSTM [91] (MAT-20)	73.64 (± 1.56)	59.53 (± 1.40)	73.31 (± 1.01)	66.50 (± 1.03)	56.27 (± 1.73)	71.98 (± 1.45)
HYBRID [91] (MAT-20)	77.95 (± 0.41)	64.52 (± 1.17)	76.35 (± 1.45)	69.17 (± 0.64)	58.75 (± 0.76)	72.17 (± 0.75)
2D-CNN [47] (DIR-784)	40.11 (± 0.56)	15.41 (± 0.82)	24.61 (± 1.18)	32.95 (± 0.65)	11.42 (± 0.62)	18.18 (± 1.06)
MLP-2 [47] (DIR-784)	27.94 (± 0.82)	4.51 (± 0.22)	8.94 (± 0.26)	21.17 (± 0.44)	4.15 (± 0.32)	8.00 (± 0.59)
MLP-1 (L7-1000)	77.76 (± 0.38)	67.85 (± 1.45)	79.75 (± 1.29)	76.11 (± 0.84)	66.95 (± 1.47)	79.63 (± 1.44)
MLP-1 (L7-784)	**78.71 (± 0.65)**	**69.79 (± 1.17)**	**81.52 (± 1.38)**	**77.16 (± 0.63)**	**67.61 (± 1.07)**	**80.11 (± 0.99)**
MLP-1 (ALL-784) ◇	96.53 (± 0.27)	94.28 (± 0.72)	96.80 (± 0.54)	97.24 (± 0.50)	95.29 (± 0.81)	97.15 (± 0.65)
MLP-1 (MAT-20) ⋆	72.54 (± 0.47)	58.29 (± 1.11)	71.87 (± 1.27)	66.94 (± 0.90)	56.51 (± 1.24)	70.88 (± 1.08)
MLP-1 (MAT-20)	64.94 (± 0.47)	48.26 (± 0.96)	63.10 (± 1.07)	54.42 (± 0.63)	40.86 (± 1.04)	57.56 (± 1.03)
Tay_RF [42] (flow-based)	84.78 (± 0.30)	75.49 (± 0.89)	83.86 (± 0.58)	80.77 (± 0.84)	72.39 (± 1.39)	81.88 (± 1.27)

"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

an unbiased input type) were shown to be not informative enough.

Fine-grained Performance via Top-K Accuracy. Delving into the performance of DL-based classifiers, in Tab. 5.5 we report their Top-K accuracy ($K \in \{1, 3, 5\}$) on the multi-class datasets.[8] From now on we exclude, for brevity, the results of DL classifiers based on biased inputs (also those including port-information). By looking at these fine-grained results, we observe that, other than the highest DL accuracy, 1D-CNN (resp. 2D-CNN) (L7-784) reports also the highest global (soft-output) behavior on the Android (resp. iOS) dataset, for example 91.51% and 93.45% (resp. 91.02% and 93.32%) accuracy when the Top-3 and Top-5 predicted apps are considered, respectively.[9] Also, although shallow (baseline) classifiers present an accuracy increase due to a larger pool of predicted apps taken into consideration, they are never able to approach the same score as the best DL classifiers, confirming also an improved global behavior of the latter (viz. learning of the TC task as a whole). This is due to shallow classifiers' inability of inferring deeply-structured traffic patterns as a whole. Indeed, they are not able to predict the true label even when considering the Top-K classes ranked by their confidence. Such "global" performance gap is even more apparent for DL classifiers resorting to packet directions, whose best Top-5 accuracy is only 68.29% (resp. 64.40%) in Android (resp. iOS) case. Hence, although mobile TC can be conceived as a conceptually-similar task to WF, it shows higher requirements with respect to the former, since the sole directions are usually sufficient for training of high-performing WF classifiers [25, 47]. Finally, the (flow-based) Tay_RF classifier provides a

[8] For the FB/FBM dataset, since the number of classes $L = 2$, we can define only the Top-1 accuracy, corresponding to the overall accuracy (see Sec. 1.3.2).

[9] Still, 1D-CNN (L7-784) performs almost on par on the iOS dataset.

Table 5.5: Top-K accuracy [%] of single-modal DL-based and baseline traffic classifiers. Results refer to the multi-class datasets and are in the format *avg. (± std.)* obtained over 10-folds. Only the classifiers fed with *unbiased* inputs are shown. **Best-performing** DL-based and shallow classifiers are highlighted for both datasets.

Architecture	*Android*			*iOS*		
	$K=1$	$K=3$	$K=5$	$K=1$	$K=3$	$K=5$
SAE [92] (L7-1000)	75.15 (± 1.52)	82.16 (± 0.85)	85.53 (± 0.72)	74.55 (± 0.80)	82.73 (± 0.92)	86.58 (± 0.79)
2D-CNN [21] (L7-784)	85.46 (± 0.48)	91.36 (± 0.31)	93.35 (± 0.30)	**82.72 (± 1.47)**	**91.02 (± 0.42)**	**93.32 (± 0.33)**
1D-CNN [94] (L7-784)	**85.70 (± 0.45)**	**91.51 (± 0.27)**	**93.45 (± 0.29)**	82.64 (± 1.63)	90.95 (± 0.36)	93.29 (± 0.32)
2D-CNN [91] (MAT-20)	76.01 (± 0.70)	86.49 (± 0.53)	90.32 (± 0.39)	68.53 (± 0.61)	82.75 (± 0.46)	87.96 (± 0.36)
LSTM [91] (MAT-20)	73.64 (± 1.56)	85.58 (± 0.58)	89.93 (± 0.50)	66.50 (± 1.03)	81.94 (± 0.88)	87.23 (± 0.73)
HYBRID [91] (MAT-20)	77.95 (± 0.41)	87.38 (± 0.37)	90.80 (± 0.29)	69.17 (± 0.64)	82.23 (± 0.38)	87.16 (± 0.39)
2D-CNN [47] (DIR-784)	40.11 (± 0.56)	58.88 (± 0.56)	68.29 (± 0.52)	32.95 (± 0.65)	53.91 (± 0.72)	64.40 (± 0.63)
MLP-2 [47] (DIR-784)	27.94 (± 0.82)	42.02 (± 0.26)	51.75 (± 0.27)	21.17 (± 0.44)	40.40 (± 0.55)	50.84 (± 0.64)
MLP-1 (L7-1000)	77.76 (± 0.38)	85.96 (± 0.30)	89.11 (± 0.20)	76.11 (± 0.84)	85.86 (± 0.65)	89.48 (± 0.51)
MLP-1 (L7-784)	**78.71 (± 0.65)**	**86.93 (± 0.40)**	**89.88 (± 0.37)**	**77.16 (± 0.63)**	**86.96 (± 0.50)**	**90.40 (± 0.51)**
MLP-1 (MAT-20)	69.94 (± 0.47)	79.22 (± 0.51)	84.94 (± 0.34)	54.42 (± 0.63)	72.47 (± 0.59)	80.03 (± 0.56)
Tay_RF [42] (flow-based)	84.78 (± 0.30)	91.69 (± 0.31)	93.89 (± 0.24)	80.78 (± 0.79)	90.70 (± 0.61)	93.58 (± 0.52)

"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

slightly better global behavior than the best single-modal DL classifier on the Android dataset, reaching 91.69% (resp. 93.89%) Top-3 (resp. Top-5) accuracy.

Training Complexity of Deep Learning Architectures. To investigate the training complexity of the considered DL classifiers, in Fig. 5.6 we report their RTPE obtained in the three datasets.[10] Results highlight a natural RTPE decrease of each classifier when the size of the classification problem is reduced (*i.e.* moving from the Android dataset, to the iOS and FB/FBM datasets). Additionally, the two classifiers reaching the highest performance are those having the highest RTPE (*i.e.* 2D-CNN (L7-784) and 1D-CNN (L7-784)), highlighting a reasonable performance-complexity trade-off. Referring to the aforementioned two classifiers, we remark that 1D-CNN (L7-784) experiences a higher RTPE than 2D-CNN (L7-784) because of lower size of the pooling layers (*i.e.* lower down-sampling) in its implementation [21, 94].

On the other hand, all DL classifiers based on "MAT-20" input present a significantly lower complexity, being this a direct consequence of the lower-dimension input set ($20 \times 4 = 80$ as opposed to 784). Analogous considerations apply to DL classifiers based on "DIR-784" input, having a lower complexity than those based on "L7-784", because the former are binary valued, with the 2D-CNN (DIR-784) showing a higher complexity with respect to MLP-2 (DIR-784), because of its more complex architecture. Finally we highlight that Fig. 5.6 reports, for the SAE, only the RTPE score corresponding to the *fine-tuning* phase (*i.e.* in which the SAE is trained in

[10]The times refer to the same hardware architecture ($8 \times$ Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz with Ubuntu 16.04 (64 bit)) in the same load conditions (*i.e.* the DL classifier is the sole CPU-intensive running process).

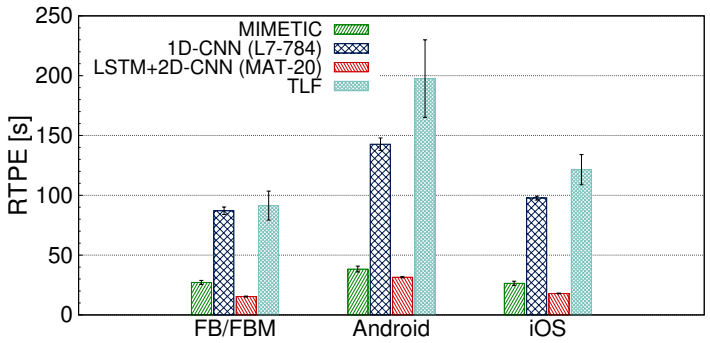


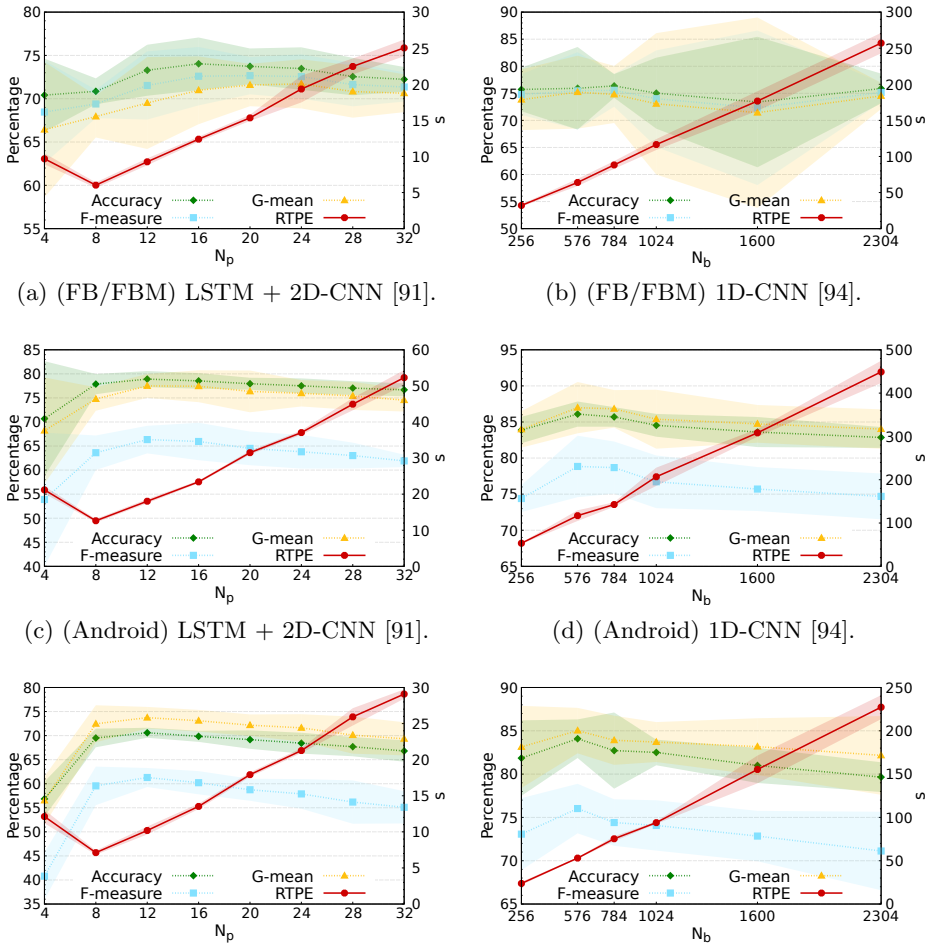
Figure 5.6: Run-Time Per Epoch (RTPE) of DL-based traffic classifiers. Results are in the format *avg. (± std.)* obtained over 10-folds. Only the classifiers fed with *unbiased* inputs are shown.

a supervised fashion as a "deep" MLP) and thus neglects its *pre-training stage*, which contributes additively to the RTPE with a linear growth in the number of AE layers (since it is done in a layer-wise fashion).[11]

Classification Performance vs. Input Size. Focusing our investigation toward the choice of the most discriminative forms of input types, in Fig. 5.7 we report accuracy, F-measure, and G-mean for the best-performing single-modal DL classifiers on the three datasets, based on two types[12] of (unbiased) input data considered herein (*i.e.* "MAT-N_p" and "L7-N_b") vs. the number of packets N_p and payload bytes N_b, varying $N_b \in \{256-2034\}$ and $N_p \in \{4-32\}$, respectively. To highlight the relevant input size-complexity trade-off, we also report the RTPE measure vs. the size of the considered input data.

[11]For example, in our scenario, the observed RTPE for the pre-training phase (of the five AE layers) equals 16.97 (±0.27) s in Android, 11.35 (±0.08) s in iOS, and 9.21 (±0.15) s in FB/FBM case.

[12]We omit, for brevity, the performance with "DIR-N_p" input, as it has been shown to be unable to reach satisfactory performance and its behavior with varying N_p can be qualitatively inferred from "MAT-N_p" results.



(a) (FB/FBM) LSTM + 2D-CNN [91].

(b) (FB/FBM) 1D-CNN [94].

(c) (Android) LSTM + 2D-CNN [91].

(d) (Android) 1D-CNN [94].

(e) (iOS) LSTM + 2D-CNN [91].

(f) (iOS) 2D-CNN [21].

Figure 5.7: Performance of the best DL-based classifier fed with *"MAT-N_p" input* (a, c, e) and *"L7-N_b" input* (b, d, f): accuracy [%], F-measure [%], G-mean [%] (left axis), and RTPE [s] (right axis) vs. *first N_p packets* (a, c, e) and *first N_b bytes* (b, d, f), for the FB/FBM (a, b), Android (c, d), and iOS (e, f) datasets. Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

From the inspection of results, it is apparent that, in the case of the N_p input (Fig. 5.7(a, c, e)), there is a *unimodal behavior* and $12 - 20$ packets are usually enough to achieve the highest performance (denoting a higher requirement with respect to the results shown in [91]), although using more packets entails an increasing of RTPE without a significant improvement in classification performance, especially in the Android and iOS scenarios.

On the other hand, in the payload size case (Fig. 5.7(b, d, f)), such trend is less obvious (although $N = 576$ bytes is observed to be the best choice among the different sizes considered), with classifiers fed with $N_b > 786$ payload bytes showing slightly poorer performance with increasing N_b values.

In both cases an *almost-linear* increase of the RTPE with the input size is apparent. The only exception is given by $N_p = 4$ packets: the reason is that, so as to implement the same DL architecture with a very small input, we had to resort to a different padding choice, implying additional complexity.

On the basis of the outcome of this analysis, we can select the best configuration of N_p and N_b values for our MIMETIC architecture, with the aim of keeping both the complexity low and also allowing an "earlier" TC. In detail, our MIMETIC instance is fed with the same unbiased input types (cf. §5.3) of single-modal DL classifiers considered herein, but with optimized (shorter) amounts of data, namely $N_p = 12$ packets and $N_b = 576$ bytes. Starting from these considerations, the next section assesses the proposed MIMETIC architecture.

5.4.2 Proposed MIMETIC-based Classifier

This section examines the performance, from both classification and complexity standpoints, of the MIMETIC approach. Building on the results of

Sec. 5.4.1, we methodically compares MIMETIC to the best existing single-modal DL alternatives, to the approaches to fuse their information (see Chapter 3), and finally to the baselines already considered in Sec. 5.4.1. We recall that the proposed implementation of MIMETIC is fed with (I) and (III) input types as prescribed in our framework, with a size of $N_b = 576$ bytes and $N_p = 12$ packets, respectively.

Overall, *four types of baselines* are included for the sake of a complete analysis:

- The *first* type of baseline is represented by the best single-modal DL classifiers fed with each of the $P = 2$ (unbiased) inputs considered in the MIMETIC approach. In the previous section, we have found that these correspond to the 1D-CNN[13] (with "L7-784" as input) [94] and the LSTM + 2D-CNN (with "MAT-20" as input) [91]. We remark that the number of payload bytes (resp. packets) used in 1D-CNN (resp. LSTM + 2D-CNN) differs from that used for MIMETIC implementation: the reason is to report performance for corresponding input-optimized versions proposed in the respective works [94, 91]. Moreover, in Fig. 5.7, classification performance does not exhibit an appreciable gain when using $N_b = 576$ bytes (resp. $N_p = 12$ packets) with respect to $N_b = 784$ byte (resp. $N_p = 20$ packets).

- The *second* type of baseline corresponds to the shallow MLP (referred to as MLP-1) already considered in Sec. 5.4.1.

- The *third* baseline is given by the flow-based RF (denoted as Tay_RF) that we have also adopted in Sec. 5.4.1.

[13]It is worth noticing that although 2D-CNN (fed with "L7-784") performs approximately on par with 1D-CNN (except on the FB/FBM dataset, see Tab. 5.3), it does not naturally consider the input as one-dimensional, thus being less meaningful.

- The *fourth* type of baseline corresponds to classifier fusion techniques, capitalizing the best single-modal DL architecture for each of the $P = 2$ inputs considered (namely, the architectures representing the first set of baselines) and combining them to get (hopefully-)improved classification results. The combination can be either performed with (simpler) non-trainable strategies, such as (*i*) Majority Voting (MV) and (*ii*) Soft-Output Average (SOA) (cf. Chapter 3). Alternatively, (*iii*) "Trainable" Late Fusion (TLF) can be pursued by concatenating the softmax layers of the two single-modal DL architectures and connecting them to a "fusion" softmax layer, in the same spirit of the "KL weights", being the best (soft) fusion rule employed in Chapter 3. In the latter case, the same training algorithm (with the same parameters) as the other DL baselines has been adopted. These combiners represent the simplest way to fuse these off-the-shelf DL traffic classifiers.

General Overview of Performance. As a high-level performance comparison, in Tabs. 5.6 and 5.7 we report the results (in terms of accuracy, F-measure, and G-mean) of the proposed MIMETIC approach, along with those of the baselines previously introduced, for the FB/FBM and multi-class datasets, respectively. First, it is apparent that the *MIMETIC architecture outperforms all the considered elements of comparison* for both metrics on all the three considered datasets, with an *improvement up to* $+8.66\%$ *and* $+7.05\%$ (*i.e.* F-measure on the iOS dataset) over the best classifier (MIOB-C) and fusion technique (MIOB-FT), respectively.

As underlined in the previous section, in the FB/FBM scenario (see Tab. 5.6), single-modal DL classifiers are able to outperform the corresponding MLP-1 (shallow) counterparts but they cannot reach the performance of

Table 5.6: Accuracy, F-measure, and G-mean [%] comparison of MIMETIC with the four groups of baselines: (I) best single-modal DL classifiers, (II) shallow neural networks, (III) state-of-the-art ML-based mobile-traffic classifier, (IV) classifier fusion techniques. Results refer to the FB/FBM dataset and are in the format *avg. (± std.)* obtained over 10-folds. The last group reports the *Maximum Improvement Over Best - Classifier* (MIOB-C) and the *Maximum Improvement Over Best - Fusion Technique* (MIOB-FT) [%] of MIMETIC architecture. Highlighted values: **overall best classifier**, best baseline classifier (⧫), and best baseline fusion technique (‡) for each dataset and performance measure.

	Architecture	*Accuracy*	*F-measure*	*G-mean*
	MIMETIC	**79.98 (± 0.49)**	**79.63 (± 0.51)**	**79.53 (± 0.60)**
I	1D-CNN [94] (L7-784)	76.37 (± 0.73)	75.56 (± 1.01)	74.79 (± 1.76)
I	HYBRID [91] (MAT-20)	74.26 (± 0.98)	73.23 (± 0.95)	72.18 (± 1.05)
II	MLP-1 (L7-784)	74.46 (± 0.88)	73.89 (± 0.86)	73.55 (± 0.89)
II	MLP-1 (MAT-20)	68.93 (± 1.32)	67.86 (± 0.94)	66.98 (± 0.75)
III	Tay_RF [42] (flow-based)	79.56 (± 0.62) ⧫	78.73 (± 0.62) ⧫	78.37 (± 0.76) ⧫
IV	MV	75.13 (± 0.92)	74.48 (± 1.14)	74.02 (± 1.65)
IV	SOA	78.86 (± 0.79) ‡	78.37 (± 1.00) ‡	78.06 (± 1.61) ‡
IV	TLF	74.61 (± 1.57)	73.60 (± 1.80)	72.59 (± 2.14)
	MIOB-C	+ 0.42 (± 0.65)	+ 0.90 (± 0.68)	+ 1.16 (± 0.99)
	MIOB-FT	+ 1.12 (± 0.89)	+ 1.26 (± 1.14)	+ 1.47 (± 1.84)

"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

(off-line) Tay_RF. We can now notice that even employing fusion techniques of single-modal DL approaches (*i.e.* SOA), these latter perform worse than Tay_RF. This may be again attributed to the fact that FB and FBM rely on several shared services. Hence, their generated traffic looks very similar. Accordingly, either the whole biflow is required to reach a confident decision or more sophisticated approaches—such as MIMETIC—are required to infer complex traffic patterns from the first packets.

On the other hand, referring to the multi-class datasets (see Tab. 5.7), we have already shown that single-modal DL architectures represent the best

Table 5.7: Accuracy, F-measure, and G-mean [%] comparison of MIMETIC with the four groups of baselines: (I) best single-modal DL classifiers, (II) shallow neural networks, (III) state-of-the-art ML-based mobile-traffic classifier, (IV) classifier fusion techniques. Results refer to the the multi-class datasets are in the format *avg. (± std.)* obtained over 10-folds. The last group reports the *Maximum Improvement Over Best - Classifier* (MIOB-C) and the *Maximum Improvement Over Best - Fusion Technique* (MIOB-FT) [%] of MIMETIC architecture. Highlighted values: **overall best classifier**, best baseline classifier (⧫), and best baseline fusion technique (‡) for each dataset and performance measure.

	Architecture	*Android*			*iOS*		
		Accuracy	*F-measure*	*G-Mean*	*Accuracy*	*F-measure*	*G-Mean*
	MIMETIC	**89.49 (± 0.32)**	**81.51 (± 0.93)**	**91.96 (± 0.95)**	**89.14 (± 0.82)**	**82.99 (± 1.14)**	**92.25 (± 0.84)**
I	1D-CNN [94] (L7-784)	85.70 (± 0.45)⧫	78.68 (± 1.20)⧫	86.82 (± 0.87)⧫	82.64 (± 1.63)⧫	74.34 (± 1.29)⧫	84.00 (± 1.31)⧫
I	HYBRID [91] (MAT-20)	77.95 (± 0.41)	64.52 (± 1.17)	76.35 (± 1.45)	69.17 (± 0.64)	58.75 (± 0.76)	72.17 (± 0.75)
II	MLP-1 (L7-784)	78.71 (± 0.65)	69.79 (± 1.17)	81.52 (± 1.38)	77.16 (± 0.63)	67.61 (± 1.07)	80.11 (± 0.99)
II	MLP-1 (MAT-20)	64.94 (± 0.47)	48.26 (± 0.96)	63.10 (± 1.07)	54.42 (± 0.63)	40.86 (± 1.04)	57.56 (± 1.03)
III	Tay_RF [42] (flow-based)	84.78 (± 0.30)	75.49 (± 0.89)	83.86 (± 0.58)	80.77 (± 0.84)	72.39 (± 1.39)	81.88 (± 1.27)
IV	MV	80.41 (± 0.40)	71.28 (± 0.85)	81.74 (± 0.77)	77.24 (± 0.62)	66.49 (± 0.97)	78.92 (± 0.97)
IV	SOA	87.08 (± 0.29)‡	80.07 (± 0.81)‡	87.00 (± 0.80)‡	84.68 (± 0.55)‡	75.94 (± 1.10)‡	84.15 (± 0.96)‡
IV	TLF	68.87 (± 1.05)	48.82 (± 1.92)	62.55 (± 1.86)	62.01 (± 0.97)	39.07 (± 1.52)	54.07 (± 1.94)
	MIOB-C	+ 3.79 (± 0.59)	+ 2.83 (± 1.66)	+ 5.14 (± 1.06)	+ 6.50 (± 2.12)	+ 8.66 (± 1.77)	+ 8.25 (± 1.72)
	MIOB-FT	+ 2.40 (± 0.48)	+ 1.44 (± 1.56)	+ 4.96 (± 1.46)	+ 4.46 (± 1.01)	+ 7.05 (± 1.43)	+ 8.10 (± 1.27)

"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

baseline (*i.e.* they outperform both MLP-1 (L7-784/MAT-20) and Tay_RF). Furthermore, improved (although similar) performance is obtained by leveraging the SOA (*i.e.* the best fusion baseline observed) of these architectures. For example, in the Android setup, 87.08% accuracy and 80.07% F-measure are achieved by SOA, as opposed to 84.78% and 75.79%, respectively, obtained by Tay_RF, and a similar reasoning applies to iOS case. However, these results are still worse than those obtained with MIMETIC approach that can outperform the SOA in all cases, attaining +4.96% (resp. +8.10%) G-mean on the Android (resp. iOS) dataset. This outcome can be directly attributed to the gain, ensured by the (multi-modal) MIMETIC architecture, arising from sophisticated fusion of the input types considered. Indeed, the latter provides a higher discriminative power in the case of very similar apps, like FB and FBM, and also further improves the effectiveness of DL in the multi-class setups.

Fine-Grained Performance. We now deepen the (classification) performance investigation of the MIMETIC framework (along with the baselines), initially by reporting their Top-K accuracy ($K \in \{1, 3, 5\}$) on the multi-class datasets in Tab. 5.8. We highlight that the above table does not include MV Top-K accuracy score, as the latter is based on classifiers' hard outputs and therefore there is no (natural) definition for the confidence vector associated to its decision. Clearly, from the inspection of these fine-grained results it is apparent that all the considered classifiers are able to improve their accuracy when a larger pool of predicted apps may be taken into consideration, (shallow) MLP-1 classifiers included. In detail, *MIMETIC* reports the *highest global accuracy* (soft-output) behavior in both the multi-class datasets, surpassing not only MLP-1 but also single-modal DL approaches. Indeed, although the (off-line) Tay_RF classifier provides a slightly better global

Table 5.8: Top-K accuracy [%] comparison of MIMETIC with the four types of baselines. Results refer to multi-class datasets and are in the format *avg. (± std.)* obtained over 10-folds. Top-K accuracy of MV is not reported due to unavailability of soft-outputs. Highlighted values: **overall best classifier**, best baseline classifier (⧫), and best baseline fusion technique (‡) for both multi-class datasets and each K considered.

	Architecture	*Android*			*iOS*		
		$K=1$	$K=3$	$K=5$	$K=1$	$K=3$	$K=5$
	MIMETIC	**89.49 (± 0.32)**	**94.29 (± 0.28)**	**95.82 (± 0.28)**	**89.14 (± 0.82)**	**95.17 (± 0.37)**	**96.74 (± 0.32)**
I	1D-CNN [94] (L7-784)	85.70 (± 0.45)⧫	91.51 (± 0.27)	93.45 (± 0.29)	82.64 (± 1.63)⧫	90.95 (± 0.36)⧫	93.29 (± 0.32)
I	HYBRID [91] (MAT-20)	75.24 (± 0.58)	85.60 (± 0.47)	89.80 (± 0.34)	70.80 (± 1.06)	83.34 (± 0.69)	87.82 (± 0.48)
II	MLP-1 (L7-784)	78.71 (± 0.65)	86.93 (± 0.40)	89.88 (± 0.37)	77.16 (± 0.63)	86.96 (± 0.50)	90.40 (± 0.51)
II	MLP-1 (MAT-20)	69.94 (± 0.47)	79.22 (± 0.51)	84.94 (± 0.34)	54.42 (± 0.63)	72.47 (± 0.59)	80.03 (± 0.56)
III	Tay_RF [42] (flow-based)	84.78 (± 0.30)	91.69 (± 0.31)⧫	93.89 (± 0.24)⧫	80.77 (± 0.84)	90.70 (± 0.61)	93.58 (± 0.52)⧫
IV	SOA	87.08 (± 0.29)‡	92.83 (± 0.31)‡	94.66 (± 0.26)‡	84.68 (± 0.55)‡	92.36 (± 0.28)‡	94.65 (± 0.23)‡
IV	TLF	68.87 (± 1.05)	79.35 (± 0.92)	83.41 (± 0.86)	62.01 (± 0.97)	75.03 (± 0.64)	80.40 (± 0.55)

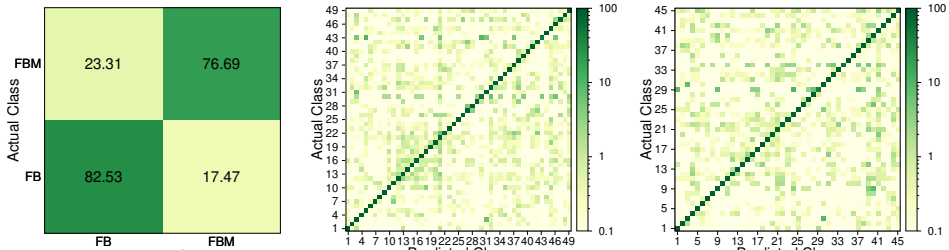
"HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

behavior than the best single-modal DL classifier (for $K \in \{3, 5\}$ on Android dataset and $K = 5$ on iOS dataset), namely 1D-CNN (L7-784), it is *able to outperform neither the proposed MIMETIC approach nor the simpler SOA combination of single-modality DL classifiers*. This result suggests that the capitalization of multiple modalities improves the global discrimination capabilities of DL-based classifiers. In detail, in Android (resp. iOS) scenario, the MIMETIC architecture is able to reach 94.29% and 95.82% (resp. 95.17% and 96.74%) accuracy when the Top-3 and Top-5 predicted apps are considered, respectively.

Then, to assess possible noteworthy misclassification-patterns and their mitigation through intermediate-fusion, Fig. 5.8 shows the confusion matrices of the MIMETIC approach (Figs. 5.8(a–c)) in the three datasets, and compares them with those of the best-performing single-modal DL approach (Figs. 5.8(d–f)) and those obtained via SOA (Figs. 5.8(g–i)). From inspection of the results, it is apparent that all the approaches achieve almost-uniform error patterns in the multi-class setups. However, analyzing FB/FBM matrix, we can notice that 1D-CNN (L7-784) exhibits poorer performance than MIMETIC and SOA. This confirms that the main error source on FB/FBM arises from the traffic similarity of the two apps and thus highlights the inadequacy of considering DL architectures (and related inputs) in a single-modal fashion. Indeed, MIMETIC clearly achieves *less-structured and milder misclassification-patterns* in all three cases considered, as opposed not only to 1D-CNN (L7-784) but also to SOA. This proves the flexibility of the information fusion provided by our framework. Equally important, the confusion matrices highlight the *appeal of cost-sensitive learning* within the MIMETIC formulation (see the definition of loss functions in Eqs. (5.2) and (5.3)), able to deal with imbalanced TC problems (which are common in the mobile context) by preventing a classification imbalance

toward the most-represented classes.

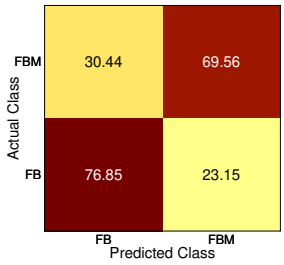
Training Complexity of Implemented MIMETIC Architecture. A useful analysis towards real-world implementations, complementing the overview of performance, is the investigation of the training complexity of the proposed multi-modal DL classifier. With this aim, Fig. 5.9a and Fig. 5.9b show, respectively, the RTPE and the overall number of trainable parameters of the implemented instance of MIMETIC architecture (considering both phases of the proposed training procedure). MIMETIC complexity is then compared against that of single-modal DL baselines when they are fed with the inputs extracted from the three datasets. For completeness, we also show the complexity of TLF baseline. Differently, the complexity of the other two classifier-fusion baselines (namely, SOA and MV) is not reported, since it is strongly linked to the training requirements of the single-modal DL classifiers being combined. Precisely, it corresponds to the more complex single-modal baseline (resp. the sum of baseline complexities) in the case of a parallel (resp. sequential) implementation. We point out that a similar reasoning applies to the pre-training phase of MIMETIC, for which the most penalizing sequential implementation of per-modality pre-training is assumed in this comparison. We highlight that the times refer to the same hardware architecture considered in Sec. 5.4.1 in the same load conditions. As expected, a decreasing RTPE is obtained when the size of the classification problem is reduced—similarly to what reported in Fig. 5.6—with a stronger trend for 1D-CNN (L7-784) and TLF, being the most complex architectures. Interestingly, MIMETIC not only reaches the highest classification performance, but it also *shows an RTPE* $> 3.5\times$ *lower* than its "main competitor" 1D-CNN (L7-784) (*i.e.* $38.34\,(\pm 0.82)$ s vs. $142.72\,(\pm 1.74)$ s) in the hardest classification (*i.e.* Android) setup. This is



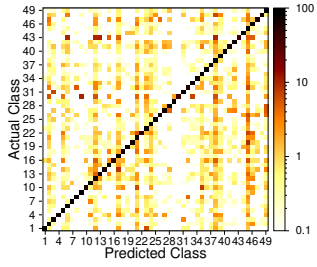
(a) (FB/FBM) MIMETIC.

(b) (Android) MIMETIC.

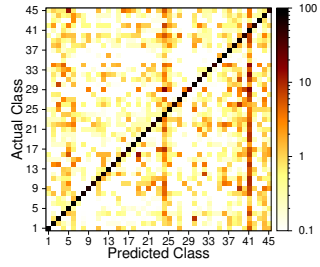
(c) (iOS) MIMETIC.



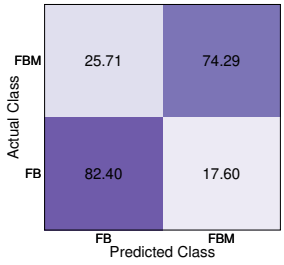
(d) (FB/FBM) 1D-CNN (L7-784).



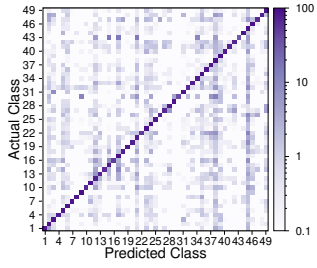
(e) (Android) 1D-CNN (L7-784).



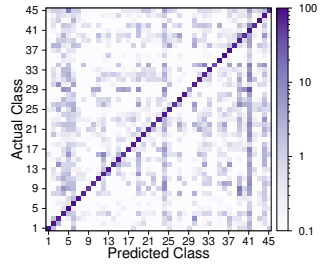
(f) (iOS) 1D-CNN (L7-784).



(g) (FB/FBM) SOA.

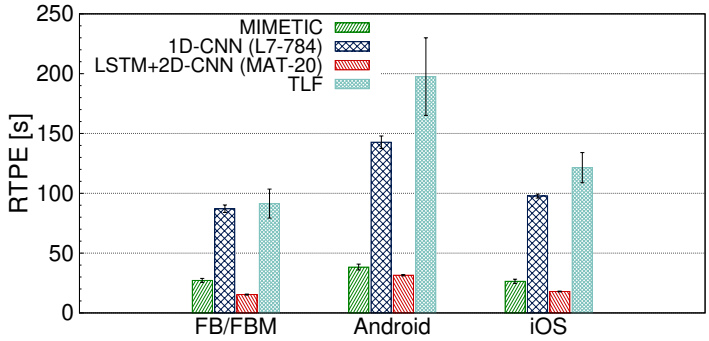


(h) (Android) SOA.



(i) (iOS) SOA.

Figure 5.8: Confusion matrices of implemented architecture based on MIMETIC (top), best single-modal DL classifier (middle), and Soft-Output Average (SOA) of single-modal DL classifiers (bottom) for the FB/FBM (a, d, g), Android (b, e, h), and iOS (c, f, i) datasets. Note that the log scale is used to evidence small errors (except for FB/FBM). Categorical class-labels for multi-class datasets are reported in Tab. 2.3.



(a) RTPE. Results are in the format *avg. (± std.)* obtained over 10-folds.

	FB/FBM	Android	iOS
MIMETIC⋆	0.9346	1.6202	1.6176
1D-CNN [94] (L7-784)	5.8223	5.8705	5.8664
HYBRID [91] (MAT-20)	0.4260	0.7380	0.7376
TLF	6.2484	6.6133	6.6081

(b) Number of trainable parameters (in millions). ⋆ numbers refer to the whole MIMETIC framework.

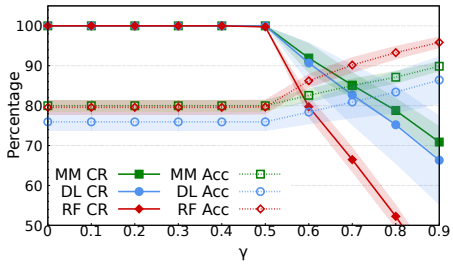
Figure 5.9: Complexity analysis. Run-Time Per Epoch (RTPE) (a) and number of trainable parameters (b) of MIMETIC architecture and DL-based baselines. "HYBRID" refers to an hybrid DL architecture combining 2D convolutional and LSTM layers (viz. LSTM + 2D-CNN) proposed in [91].

due mainly to shorter inputs and simpler (viz. computationally-lighter) layers involved in the MIMETIC instance (cf. §5.3.3), allowed by an improved capitalization of the inputs available. It is worth noting that the same reasoning equally applies with respect to MV and SOA baselines, whose complexities are dominated (in the best case) by the most complex single-modal DL architecture being fused. Moreover, MIMETIC shows also the lowest complexity increase when passing to a harder classification problem (*i.e.* it exhibits a higher scalability), being highly desirable in mobile contexts. In-

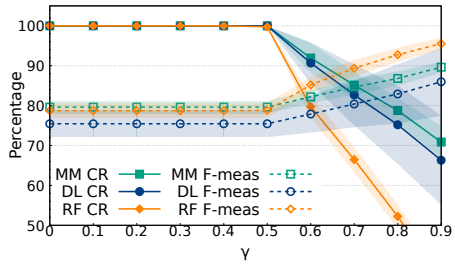
deed, a +41% increment in RTPE (against +64% for 1D-CNN, +105% for LSTM+2D-CNN, and +116% for TLF) is observed when moving from the FB/FBM to the Android dataset. Finally, the inspection of Fig. 5.9b highlights that the RTPE is strongly related to the number of parameters to be trained, with the MIMETIC approach (in its complete architectural configuration) having $\approx 3.6\times$ and $\approx 4.1\times$ fewer trainable parameters than 1D-CNN and TLF, respectively.

Performance with Reject Option. As a complementary analysis suggested in our performance workbench and oriented to finer performance control, Fig. 5.10 shows the accuracy and F-measure (first and second column of plots, respectively) of (*i*) the proposed MIMETIC approach, (*ii*) the best single-modal DL approach (*i.e.* 1D-CNN (L7-784)), and (*iii*) the flow-based Tay_RF [42] (*i.e.* the current ML-based state-of-the-art traffic classifier) vs. the censoring threshold γ on each of the three datasets. We exclude herein the SOA and shallow (*i.e.* MLP-1) approaches, due to lower (overall and fine-grained) performance (see Tabs. 5.6, 5.7, and 5.8) with respect to MIMETIC (while having a common "fusion" rationale) and single-modal DL classifiers, respectively. In detail, this analysis delves into the possibility for MIMETIC (and considered baselines) to classify apps more accurately only from reliably-labeled biflows. We notice that a threshold value implying varying performance with respect to unclassified samples, can be observed only if $\gamma \geq 1/L$—recall that L corresponds to the number of classes.[14] In all the plots, for the sake of a thorough comparison, we also report the ratio of classified samples (CR) vs. γ, for $\gamma \geq 50\%$, being smaller values of little practical interest. We highlight that we opted

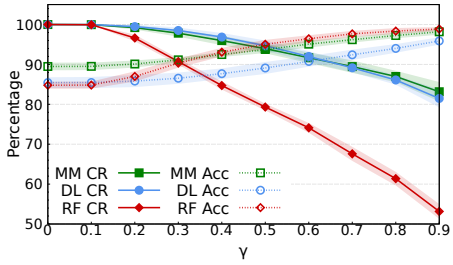
[14]Specifically, this value corresponds to 0.5 in the case of the FB/FBM dataset, whereas it equals ≈ 0.02 for Android and iOS datasets.



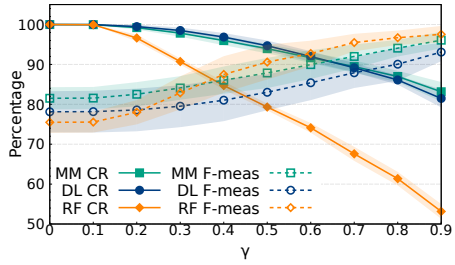
(a) (FB/FBM) Accuracy.



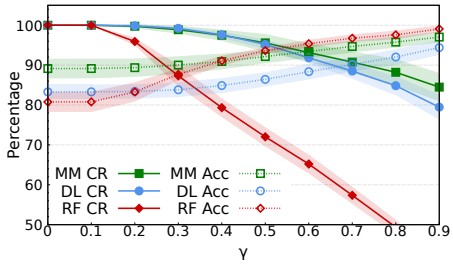
(b) (FB/FBM) F-measure.



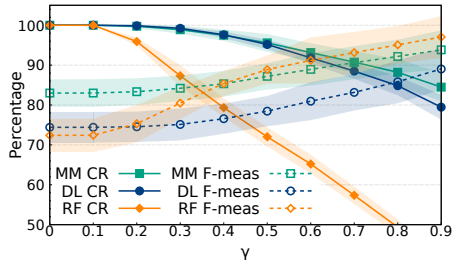
(c) (Android) Accuracy.



(d) (Android) F-measure.



(e) (iOS) Accuracy.



(f) (iOS) F-measure.

Figure 5.10: Accuracy (a-c), F-measure (d-f), and ratio of classified samples (CR) [%] vs. censoring threshold γ of MIMETIC (MM) architecture vs. best single-modal DL classifier (DL) and state-of-the-art ML-based mobile-traffic classifier (RF). Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown. To ease direct comparison, the y-axes are limited to percentages over 50%: only CR of RF continues dropping below that threshold, to values of little practical interest.

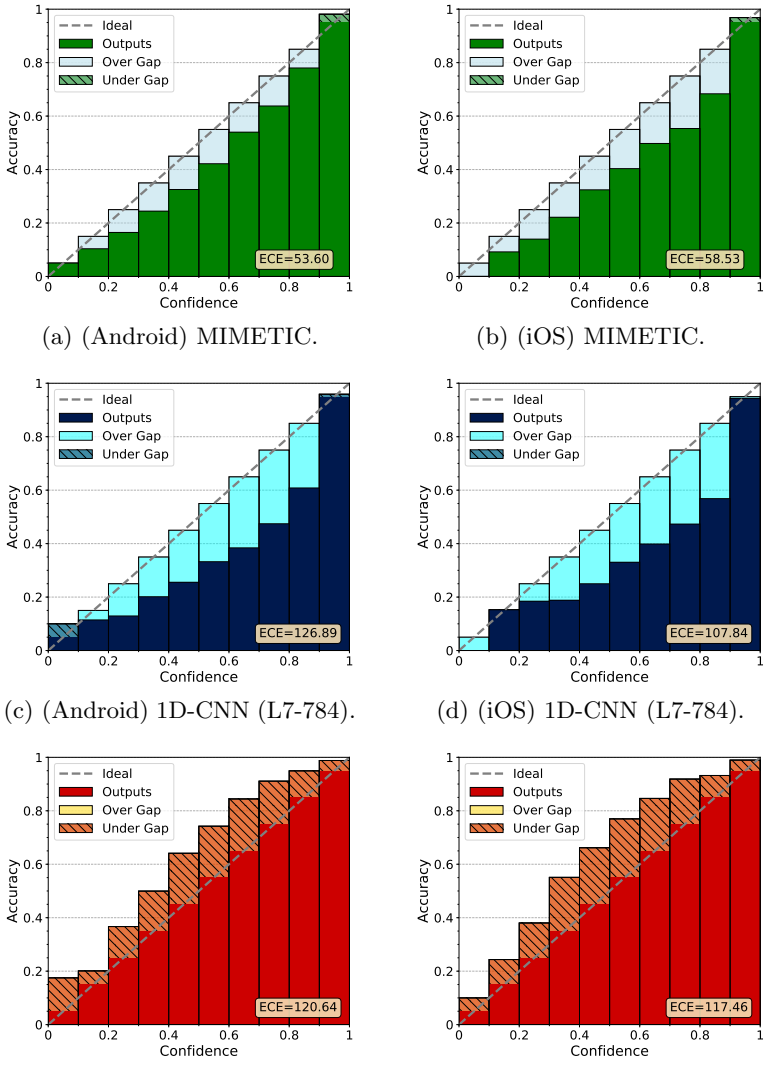
to keep the visualization of accuracy/F-measure and CR separate so as to provide a finer understanding of the corresponding interplay among the two conflicting measures. Also, to avoid redundancy, we have omitted G-mean since it shows similar trends to accuracy and F-measure, having almost the same profile (vs. γ) for all the approaches considered.

By looking at the qualitative profiles of CR and both performance measures, the following considerations can be drawn. First, the results highlight that all the methods enjoy improved classification performance when increasing γ, at the price of a decreasing CR. However, only in the multi-class dataset it is evident a relevant performance improvement with a negligible ratio of unclassified samples, whereas for the FB/FBM (binary) dataset this trend is sharper and less advantageous. Secondly, the Tay_RF approach shows the least evident improvement of accuracy and F-measure with γ which is paid, unfortunately, with a quick CR decrease. This outcome may be explained with a high number of biflows classified with low confidence by Tay_RF (see the next "calibration analysis" for details). As an example, by looking at the Android dataset and having a 90% F-measure as a target, the *Tay_RF* approach *rejects double of the tested biflows with respect to the proposed MIMETIC approach* (20% vs. 10%, respectively). On the other hand, comparing MIMETIC with the (best) single-modal DL architecture, a similar behavior of the CR with γ is observed for the two approaches, whereas MIMETIC provides an almost-constant performance improvement over all the range, also in the less-advantageous FB/FBM scenario. This again confirms the global performance gain originating from the adoption of more sophisticated multi-modal DL in our proposal. Specifically, by *rejecting* the classification of only 10% *of instances*, on the Android dataset, the proposed *MIMETIC approach* is able to achieve *accuracy and F-measure such that* $\geq 95\%$ *and* $\geq 90\%$, respectively, corresponding to an increment

$\geq$ 5% accuracy and $\geq$ 8% F-measure with respect to 1D-CNN (L7-784) and Tay_RF. Similarly, for the iOS dataset, the proposed approach is able to achieve (roughly) the same targeted performance, with also a more substantial improvement over both baselines (up to +10% and +15% F-measure against 1D-CNN (L7-784) and Tay_RF, respectively). Unfortunately, in the FB/FBM scenario, achieving $\geq$ 90% target performance on both measures would require $\approx$ 30% biflows to be censored. This result reflects the difficulty (although mitigated by MIMETIC) in solving an "overlapped-apps" classification task, sharing many third-party (common) services in their execution.

Calibration Analysis. Finally, to deepen the soft-output behavior of the mobile traffic classifiers taken into account, we complement the investigation of performance vs. reject option, with a calibration analysis as recommended in Sec. 5.2.4. To this end, Fig. 5.11 reports the *reliability diagrams* and the *ECE* of the proposed MIMETIC architecture, along with the best single-modal DL and Tay_RF baselines (*i.e.* the ones already considered in the aforementioned analysis) for the Android (left column) and iOS (right column) datasets. In this case, we do not report the performance for the FB/FBM dataset, as similar trends have been observed also in this scenario. Additionally, since it is a binary dataset, the dynamic of reliability diagrams is reduced since the class prediction probability is always higher than 0.5.

It can be seen that the MIMETIC classifier results to be better calibrated with respect to the two considered baselines, resulting in an ECE being *less than half* of that of the 1D-CNN (L7-784) and Tay_RF. This applies to both datasets. Furthermore, by looking at the behavior of each classifier on the two multi-class datasets, we can observe (i) an invariance of the



(a) (Android) MIMETIC. (b) (iOS) MIMETIC.

(c) (Android) 1D-CNN (L7-784). (d) (iOS) 1D-CNN (L7-784).

(e) (Android) Tay_RF. (f) (iOS) Tay_RF.

Figure 5.11: Reliability diagrams of MIMETIC (a & b), best single-modal DL (c & d), and Tay_RF (e & f) mobile traffic classifiers trained on the Android (a, c, e) and iOS (b, d, f) datasets. Bin width is $M = 10$. Under and over gap represent an under-confident (pessimistic) and over-confident (optimistic) miscalibration pattern, respectively.

miscalibration pattern and (*ii*) a different ECE trend.

Specifically, referring to point (*i*), both the DL-based classifiers (namely, MIMETIC and 1D-CNN (L7-784)) interestingly exhibit almost always (except for the last bin) a miscalibration that tends to be over-confident (optimistic) in its predictions (*i.e.* in each bin the confidence is higher than the accuracy). This effect can be attributed to a slight overfitting phenomenon and is one of the distinctive characteristics of DL architectures [172] (although MIMETIC mitigates it). Differently, the Tay_RF classifier shows an accuracy always higher than the related confidence, which can be attributed to a slight bias due to its "ensemble" nature (*i.e.* it is Random Forest whose decision is taken based on the average of multiple parallel decision trees) and thus validating its low-confident profile (vs. γ) shown in Fig. 5.10.

On the other hand, referring to point (*ii*), MIMETIC performs better on the Android dataset, whereas the two baselines are more effective on the iOS one. However, a relative performance inversion is observed between the two baselines when passing from the Android to the iOS dataset, namely Tay_RF performs better than the 1D-CNN (L7-784) on Android, whereas the 1D-CNN (L7-784) outperforms Tay_RF on iOS. This confirms that the difference in software and hardware ecosystems of these mobile operating systems impacts also on the traffic proprieties and consequently on the app discrimination ability of mobile traffic classifiers, being not generalizable between the two cases.

Final Remarks. The analysis performed in this chapter and based on the proposed framework enables the design, evaluation, and comparison of effective (mobile) traffic classifiers via DL, with the aim to avoid pitfalls and be the springboard of real-world implementations. Indeed, the formalized evaluation workbench allows to investigate these classifiers both at a

finer detail (*e.g.*, considering per-app and fine-grained measure or reject-option and calibration analysis) and also taking into account their complexity, representing a key aspect in mobile TC where frequent re-training of a classifier is required, due to aging of training data (*e.g.*, because of apps/OS updates). Moreover, we have shown that mobile TC presents its own peculiarities, which hinder the straightforward application of DL classifiers originated from other domains (*e.g.*, image/speech processing). Specifically, skimming informative and unbiased information from traffic data used to feed the DL classifiers is essential to prevent misleading performance results.

Nevertheless, based on the results of this careful evaluation, we notice that there is no "killer" DL architecture for mobile TC. Indeed, the most the DL model fits the nature of the input data, the better it is expected to perform (one relevant example is the comparison of 1D- and 2D-CNN based on payload data which is, by definition, one-dimensional). From these observations we derive that, given the heterogeneous information available from traffic data, the need for advanced DL architectures arises. To this end, we propose MIMETIC, a multi-modal DL approach able to capitalize heterogeneous input data by capturing intra- and inter-modal dependencies and implement a specific instance of its general architecture. Leveraging the proposed framework, we show that MIMETIC is able to outperform both ML- and DL-based baselines, while having a training time more than three times lower than its main single-modal DL competitor.

Chapter 6

Big Data-Enabled Traffic Classification

Chapter 5 has shown that Deep Learning (DL) represents a promising solution toward the fulfillment of high performance in the dynamic and challenging (mobile and encrypted) TC context and has attracted the attention of research community, with several works recently appeared tackling TC via DL (see Tab. 5.1). Nevertheless, our comprehensive investigation has revealed that DL-based TC resulted thorny, and generally less well understood than that adopting approaches based on standard ML. Indeed, DL algorithms may generate learning networks with a very dense and complex structure [68], whose training may result in completion times orders-of-magnitude higher than those acceptable according to the constraints of the specific application domain.

The constant repetition of tasks requiring high computational power and strict time constraints is the target of Big Data (BD) frameworks. Hence, leveraging BD parallelization potential is sought to be a solution to DL-based TC. However, although BD framework embodies a transparent accelerator to *separable* computational tasks (*e.g.*, the test phase of inference

systems), this is not the case for non-naturally-parallelizable ones, like the optimization in DL training procedure [173].

This motivated our research, in which *for the first time in literature we investigate and experimentally evaluate the adoption of DL-based network traffic classification strategies as supported by BD frameworks.* In more details, pursuing our analysis along three different (but inter-playing) dimensions—*i.e.* classification performance, training completion time, and costs—in this chapter, we design, deploy, and evaluate state-of-art DL networks (namely, 1D-CNN and LSTM) for classifying encrypted mobile traffic via BD. In our experimental campaigns, we run classification tasks adopting the BD platform of a public-cloud service provider and still leveraging the common benchmark described in Sec. 2.5.1, encompassing human-generated mobile and encrypted traffic datasets. This choice provides also results related to popular and reproducible setups as well as to real-world traffic. Accordingly, this investigation is able to deliver a picture detailed at a depth never achieved before, producing interesting outcomes and useful guidelines for both researchers and practitioners willing to harvest the benefits deriving from the *joint* adoption of DL and BD in network traffic analysis, also complementing, from an architectural viewpoint, the set of methodologies for mobile and encrypted TC proposed in this Thesis.[1]

The rest of this chapter is organized as follows. Sec. 6.1 briefly reviews the existing literature on BD network analytics (including traffic analysis and classification); Sec. 6.2 illustrates the reference BD-enabled DL framework for mobile TC, focusing on key aspects pertaining to the design phase; Sec. 6.3 describes the experimental evaluation setup considered, with corre-

[1] We highlight that the BD accelerator can be applied also in conjunction with ML traffic classifiers. However, its benefits are way fruitful in the case of computationally-hungrier DL-based approaches.

sponding results discussed in Sec. 6.4.

6.1 Related Works

In this section, we position our contribution against the available BD-based solutions to address networking issues, that is not limited to traffic analysis and identification/classification.

In Tab. 3.1, we have shown that various works tackled mobile TC in recent years, mostly via standard ML techniques and often under encrypted-traffic assumption. Also, in Tab. 5.1, we have reported a number of proposals that lately emerged proving the appeal of DL to Internet TC (and conceptually-related WF). However, for the latter only initial design attempts are provided, all related to either *non-mobile* or *non-encrypted* scenarios, being the proposal of Chapter 5 the first methodological approach to design DL-based mobile and encrypted traffic classifiers.

On the other hand, in line with the interest of the scientific community, many works have employed *BD solutions* in the broad field of networking to capitalize the value of network data, notwithstanding the constraints they impose. These works mostly fall in the area of either network security [174, 175, 176], or mobile and social networks analytics [177, 178], and (almost) all benefit from distributed computations aimed at reducing the time required for training ML models. Instead, only a few works specifically leverage BD solutions to focus on *network TC via ML* [97, 79, 87, 95], with only [97] tackling the mobile case (see Tab. 1.2). D'Alessandro *et al.* [79] implement a distributed SVMs framework for Internet TC adopting Hadoop[2] to design a global parameter store that maintains the shared parameters during the training phase of SVMs. Using a cluster of up to

[2]http://hadoop.apache.org/

20 nodes, the authors claim that the training process of their solution is up to 9× faster than standalone SVM. Similarly, Yuan and Wang [87] devise *HAC4.5*, a parallelized version of common C4.5 based on the Hadoop file-system and MapReduce parallel-processing framework, and employing six cluster nodes. Leveraging the Moore dataset [106], the experimental results show only a minimal improvement over a non-optimized C4.5. Ke *et al.* [95] integrate the Spark framework into their feature selection algorithm for Internet TC. Specifically, they employ growth algorithm to discover the relevant (viz. direct correlation) features and shrink algorithm to eliminate redundant weak correlation ones, using Spark[3] (with one, two, and four nodes) to improve the efficiency of the algorithm. Here too, the authors use the Moore dataset for performance evaluation, reporting an increasing computational advantage of multi-nodes setup with the expansion of data scale. As already mentioned in Sec. 3.1, Le *et al.* [97] envision a distributed computing platform to integrate various ML algorithms, BD analytics platforms (*e.g.*, Apache Spark, IBM InfoSphere, etc.), software-defined networking, and network functions virtualization for 5G self-organizing network applications. As a proof of concept, the authors implement an online classification model in the InfoSphere cluster[4], consisting of one computing master and four workers. They show that the maximum classification speed is around 90k biflows/second with a latency ≈ 10ms, on average.

Recently, a few frameworks have bloomed for leveraging BD infrastructures to train (and run) DL algorithms in different flavors. However, only a very limited set of works has already adopted BD for addressing networking issues through DL algorithms [178, 179]. Alsheikh *et al.* [178] focus on an activity recognition based on mobile-device data and evaluate the proposed

[3]https://spark.apache.org/
[4]https://www.ibm.com/analytics/information-server/

setup in terms of both speedup efficiency and accuracy. Differently, Abeshu and Chilamkurti [179] envisage DL adoption in fog-to-things communication scenario for attack detection. Nonetheless, all these works mainly focused on how BD frameworks are able to reduce the completion time of the DL heavy tasks and—to the best of our knowledge—none of them evaluated the detrimental effect of distributing data-analysis tasks across several (loosely coordinated) workers.

To the best of author's knowledge, (i) no work has performed TC by means of BD-enabled DL classifiers to date. Equally important, (ii) the challenging scenario of encrypted mobile traffic data has been only touched tangentially within the BD framework, even considering (classic) ML techniques. Finally, (iii) the validation leveraging *human-generated traffic*—that is of paramount importance toward real-world implementations in mobile contexts—has been often overlooked.

6.2 Big Data-enabled Deep Learning-based Mobile Traffic Classification

A basic scheme for the proposed BD-enabled DL-based mobile TC solution is reported in Fig. 6.1. Its related design choices can be categorized in those strictly concerning the TC workflow (that are BD-independent) and those related to the training mechanisms enforced by the DL architectures when deployed on a BD framework (that are BD-dependent, by definition). We refresh the former in Sec. 6.2.1, while we discuss in detail the latter in Sec. 6.2.2.

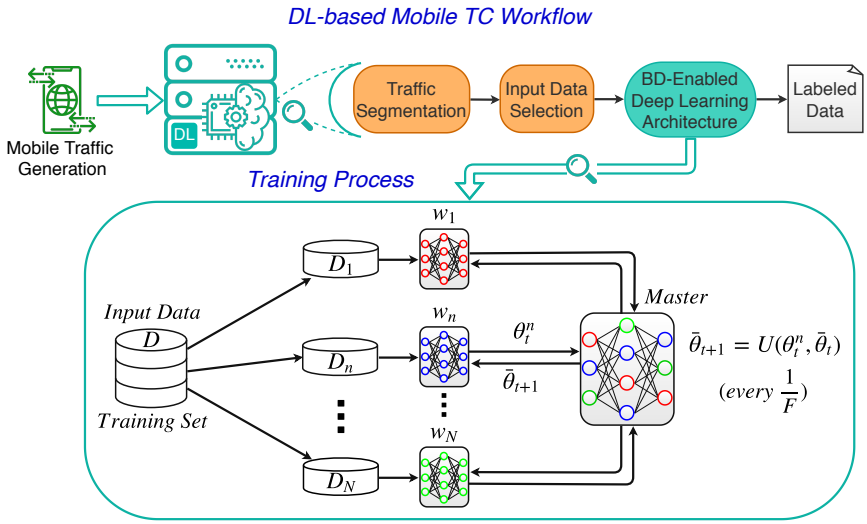


Figure 6.1: Scheme for the proposed BD-enabled DL mobile TC solution.

6.2.1 Deep Learning-based Mobile Traffic Classification Workflow

As prescribed by the framework devised in Sec. 5.2, to design a DL architecture for TC, milestone design choices should be made about: (i) the *traffic object* (*i.e.* the traffic aggregate atom which induces the segmentation criterion); (ii) the *type(s) of input data*, (*i.e.* the number and the sets of input selected from each traffic object to feed the DL architecture); (iii) the *DL architecture* (*i.e.* the composition instance of elementary learning layers) coping with input constraints originating from the design choices concerning the *type of input data*. We briefly discuss these aspects regarding the proposed BD-enabled solution in the following, pointing to Sec. 5.2 for a more detailed analysis.

Appealing *traffic objects* that might be adopted are the *flow* and *biflow*—the latter also leading to better performance than the unidirectional one—

and those purposely defined for mobile TC (*e.g.*, service bursts), even tough these last have some limitations as reported in Sec. 5.2.1.

The next step after segmentation is to extract for each traffic object the corresponding unbiased *input set(s)*, especially those suited for "early" TC. In Sec. 5.2.2, we have shown that the most relevant unbiased types of input data of a generic traffic object ingested by DL architectures may be roughly grouped within two categories: (i) the first N_b bytes of the payload at transport level or higher; (ii) selected informative data fields of the first N_p packets. In the first case, the payload data being fed to the DL architecture is represented in binary format, arranged in a byte-wise fashion and normalized so as to constrain it within $[0, 1]$. In the second case, the type of input data is represented by selected protocol fields not pertaining to the explicit inspection of encrypted payload (*e.g.*, the packet size) of the first N_p packets.

Finally, the *DL architectures* are topped with a softmax layer providing inference among L possible apps, and are obtained by composition of elementary layers, whose common choices are *dense*, *convolutional*, *pooling*, and *recurrent layers*, as depicted in Fig. 5.2 and described in Sec. 5.2.3.

6.2.2 Training Deep Learning-based Mobile Traffic Classification Architectures on Big Data

The learning process for DL architectures may be slow and computationally demanding, since they consist of many hidden layers, millions of parameters and require a high number of training samples. BD solutions are meant to offer a way to address these issues, providing processing frameworks able to parallelize computational tasks by splitting the information base and distributing it across N *cooperating working nodes* (*workers*) coordinated by a single central node (*master*).

Specifically, BD-enabled DL relies on *data parallelism* and *federated learning* [173] to reduce the overall training time of the considered DL architecture, by capitalizing the peculiarity of BD paradigm. In essence, the workers $w_1, \ldots, w_N$ are given N distinct partitions $\mathcal{D}_1, \ldots, \mathcal{D}_N$ of the training set $\mathcal{D}$ to learn independent replicas of the (same) given DL architecture. Clearly, deploying a higher number of workers allows to enhance the parallelization, namely the higher N, the smaller the size of the partitions $\mathcal{D}_1, \ldots, \mathcal{D}_N$ assigned to the workers. On the other hand, each worker is able to learn only a "data-partial" DL model, being the outcome of its limited-view training partition, in principle. Additionally, since learning is based on (sophisticated versions of) stochastic gradient descent, the process of the n^{th} worker is naturally iterative and performed over N_{epo} "epochs", composed of different mini-batches (scanning the whole $\mathcal{D}_n$), with the model at time t completely specified by the parameter set $\boldsymbol{\theta}_t^n$.

In federated learning, different workers are federated by the master to optimize a central DL model—specified at time t by the parameter set $\bar{\boldsymbol{\theta}}_t$—exploiting their DL model replicas by minimizing a single (common) loss function $\mathcal{L}(\cdot)$, being for TC a categorical cross-entropy [68], and implicitly capitalizing the whole training set $\mathcal{D}$. This is achieved by *periodically synchronizing* the state of each worker with the (centralized) view of the master, whose model is incrementally updated leveraging the information provided by the workers. The master is in charge of the coordination mechanism and has the responsibility to incorporate model updates periodically coming from the workers (*worker commits*), and to serve the requests of the most updated central model (*worker pulls*). Between subsequent commits each worker learns *independently* on its training partition. The *worker update frequency* F at which the workers execute a commit is thus a design parameter. Such frequency ranges from one update per mini-batch to

exchanges after several epochs, the higher (resp. lower) values leading to tighter (resp. looser) coupling.

Additionally, depending on the *communication protocol* governing the exchange of commits/pulls between the workers and master, BD-enabled DL approaches can be categorized into two main groups: *synchronous* and *asynchronous*. In the former case, the commits from the workers are aligned through a synchronization barrier, and the pull operation puts all the nodes in the same state $\bar{\boldsymbol{\theta}}_{t+1}$ after the master aggregation. In the latter case, the commits from the workers are handled in a first-come first-served fashion by the master, which provides the updated central model $\bar{\boldsymbol{\theta}}_{t+1}$ based on the message from the worker. Although the latter solution can incur the side-effect that some workers are computing (and committing) updates based on old central-model states (since the master incorporates updates into the central model asynchronously), it is more time-efficient because it does not include locking mechanisms that make all the workers wait for the slowest one (the so-called *straggler* issue) and works well also with heterogeneous hardware.

Lastly, the *federated-optimization algorithm* is another degree of freedom of the BD DL-based TC system proposed. It is defined by both local workers' computation and master update policy and is tightly coupled to the communication protocol choice. Precisely, for each update of the central model, in the synchronous (resp. asynchronous) case the master uses all the commits at once (resp. one commit at a time).

Accordingly, the adoption of BD framework to support the learning process of DL architectures is expected to greatly reduce the time required for its training on the *whole* $\mathcal{D}$. However this benefit comes at a cost: since no node has the chance of working on the whole dataset, the DL architecture resulting from this training procedure represents a *sub-optimal solution* to

the TC problem, exposing performance possibly worse than that of a centralized solution (with much longer processing periods but working on the $\mathcal{D}$ training set as a whole). Hence, next section investigates the dependence of DL training in mobile TC on the non-transparent BD accelerator.

6.3 Evaluation Setup

In this section, we detail the setup designed and adopted for the experimental evaluation. We resort to the three datasets being part of the common benchmark described in Sec. 2.5.1 and already employed for the assessment of (not BD-enabled) DL-based solutions carried out in Sec. 5.4. Since they are associated to different mobile and encrypted TC tasks (*e.g.*, FB/FBM dataset to billing differentiation and multi-class ones to service prioritization), we leverage them to understand if and how the BD infrastructure impacts the performance of the different mobile TC problems. Also, coherently with the observations reported in Sec. 5.2.1 and for the sake of a consistent assessment of almost all DL-based TC works published so far (see Tab. 5.1), we have chosen to operate at the *biflow* level.

Turning to the details, in Sec. 6.3.1, we specify the BD-enabled DL-based TC architecture deployed and the tools we adopted. Finally, in Sec. 6.3.2, we introduce the performance metrics to investigate the proposed TC approach along different dimensions induced by BD solutions.

6.3.1 Architecture Deployment

Herein we describe the experimental setup designed and implemented to evaluate the performance of the DL-based TC solutions when deployed onto BD architectures.

In line with the strategies usually adopted today by enterprises aim-

ing at achieving both technical and economical advantages, we run all our experimental-evaluation campaigns onto a cloud platform. In detail, we utilize the services of *Microsoft Azure*, one of the market leaders among the cloud providers. The impact of this decision on our analysis is two-fold: (*i*) some of the following deployment choices depend upon the options commercialized by the provider; (*ii*) the adoption of a public-cloud platform puts under the spotlight the *economical expenditure* generated by the execution of DL tasks. Though this choice may place constraints on the experimental analysis because of the finite budget available, it allows us to further enrich our study with interesting results along dimensions other than classification performance, such as the *cost* charged to cloud customers for accomplishing model training tasks (see §6.3.2).

We have obtained all the results discussed in Sec. 6.4 leveraging *Distributed Keras* [173], a distributed DL framework built on top of Apache Spark[3] and Keras [170]. In details, we rely on *Azure Databricks*[5], which provides analytic services based on an Apache Spark environment optimized for DL. Distributed Keras provides several state-of-art optimization algorithms (based on data-parallelism and federated learning) and is claimed to reduce the time spent for training models with respect to traditional centralized approaches.

Specifically, we have selected the inputs for the experimental setup (*i.e.* number of workers N and worker update frequency F) according to budget constraints as well as observed trends, so as to explore satisfactorily the space generated by all their combinations. In detail, we consider deployments with the number of workers ranging from $N = 2$ to $N = 16$, while for F we have considered values from one update per mini-batch (*i.e.* ≈ 139 up-

[5]https://azure.microsoft.com/it-it/services/databricks/.

dates per epoch in our experimentation) to one update every N_{epo} epochs (*i.e.* one single update per worker).

Furthermore, we have chosen the setup of master and worker nodes according to the offers of the cloud provider, by adhering to the default setting which employs the same node configuration for both the master and the workers. In detail, we use general-purpose `DS4v2` nodes (8 vCPUs, 28 GiB RAM, 0.698 €/hour) in all our experiments, with better-performing `D32sv3` nodes (32 vCPUs, 128 GiB RAM, 2.456 €/hour) leveraged for specific analyses, as detailed later.

Finally, we have selected the DL architectures based on diversity of elementary layers and considering those attaining the best performance (for each unbiased input type, see Sec. 6.2.1) in a centralized deployment (see §5.4): a 1D-CNN [94] (fed with the first $N_b = 784$ payload bytes of the transport level) and an LSTM [91] (fed with four informative fields[6], of the first $N_p = 20$ packets in a biflow). The former corresponds to 5.82M, 5.87M, and 5.86M, while the latter to 52.3k, 57.1k, and 56.6k *training parameters* for FB/FBM, Android, and iOS datasets, respectively. Concerning the optimization algorithm, we adopt the *Asynchronous Elastic Averaging Stochastic Gradient Descent (AEASGD)* with $N_{\text{epo}} = 90$ [173], being asynchronous and thus able to avoid the straggler issue.

6.3.2 Evaluation Metrics

Here we introduce the metrics adopted to evaluate the DL architectures when deployed on cloud BD frameworks. Consistently with the previous chapters, our experimental analysis resorts to a stable performance-evaluation setup, based on a stratified ten-fold cross-validation. Hence, for

[6]Packet size, packet direction, TCP window size, inter-arrival time.

each of the metrics discussed in what follows, we report its mean and standard deviation. Notably, our experimental evaluation is performed along three distinct dimensions: (*i*) *training completion time*, (*ii*) *cloud deployment cost*, and (*iii*) *classification effectiveness*. We are interested in investigating the *trade-offs* existing among these three intertwined dimensions. The metrics defined and adopted for each dimension are detailed in the following.

Training Completion Time. Since reducing the processing time required for a task completion is arguably the major driver leading to the adoption of BD architectures, we provide a detailed evaluation of this key aspect, focusing on the *wallclock time* T required for completing the *training phase* of DL architectures.[7] We recall that this analysis is of great interest since mobile TC systems require frequent re-training operations, due to aging of training data as a result of both app and OS updates. Precisely, since (distributed) DL training is performed on multiple epochs [68]—similarly to Chapter 5, but referred to a cloud scenario—we report such information in a normalized way, as *Wallclock Time Per-Epoch (WTPE)*.

Cloud Deployment Cost. Cloud services are characterized by pay-as-you-go billing strategies, thus abolishing capital expenditure for configuring and maintain the BD infrastructure. Accordingly, here we consider the total cost C charged to the cloud customers for running the processing tasks needed for training the DL architecture. Specifically, our *cost evaluation function* is:

$$C = (\rho\, N + \rho_M)\, T \tag{6.1}$$

[7]We recall that time reduction trends of testing phase are less interesting, due to perfect parallelization.

where N denotes the number of workers, ρ (resp. ρ_M) the hourly cost for deploying one worker node (resp. the master), and T the training completion (viz. wallclock) time.

Classification Effectiveness. Because BD frameworks do not represent a transparent accelerator for the training phase of DL-based traffic classifiers, to evaluate the effectiveness of the corresponding DL-based TC solutions, the adopted evaluation metrics include common classification measures (cf. §1.3.2) such as the macro *recall* and *F-measure*, as well as *confusion matrices* to identify the most frequent misclassification patterns.

6.4 Experimental Evaluation

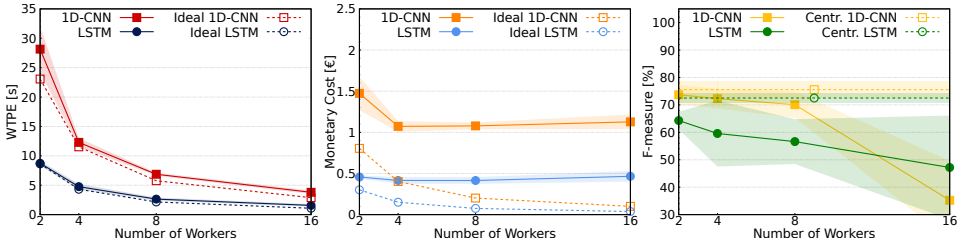
Herein we discuss the results of the experimental campaigns we run deploying the designed system on Azure PaaS to evaluate its performance against the mobile TC tasks related to three different datasets (*i.e.* binary and multi-class), along the three evaluation dimensions (*i.e.* completion time, monetary cost, and classification effectiveness, see §6.3). For each of these, we assess the impact of different design choices such as the number of workers (N), the update frequency (F), and the DL architecture.

Completion Time vs. Number of Workers (N). Figs. 6.2a, 6.2d, and 6.2g show the WTPE for the two considered DL-based TC architectures on FB/FBM, Android, and iOS datasets, respectively, when increasing N from 2 to 16. Herein, the worker update frequency F is set to one update per epoch. To stress the overhead incurred by each BD-enabled DL architecture, we consider the corresponding WTPE T_1 needed to run it in a centralized fashion, namely when *one worker* is in charge of processing the

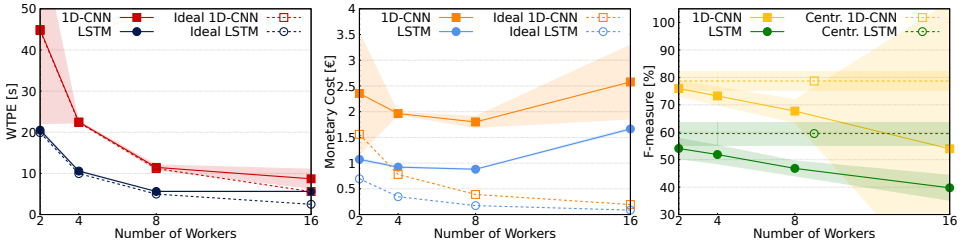
whole training set. Accordingly, we report the *ideal-WTPE curve*, defined as T_1/N and corresponding to a *lower-bound* on the achievable WTPE.

The results show an intuitive decreasing trend with N for all TC tasks (with slightly higher WTPE for the multi-class datasets, in line with the more complex classification tasks), thus confirming the appeal of the BD framework which is able to reduce the training time up to -91.8%, -90.3%, and -88.5%, when a 1D-CNN is used in the case of FB/FBM, Android, and iOS, respectively, with respect to an analogous centralized deployment. For example, with $N = 8$ workers, $\leq 11s$ WTPE is required in all TC tasks. Additionally, the overhead incurred with respect to the theoretical curve also increases for higher values of N (*i.e.* the larger N, the higher the overhead), but remains negligible, although more evident in the multi-class scenarios. Finally, a direct comparison of the two different DL architectures shows that the more complex 1D-CNN benefits more from parallelization with respect to the "lighter" LSTM.

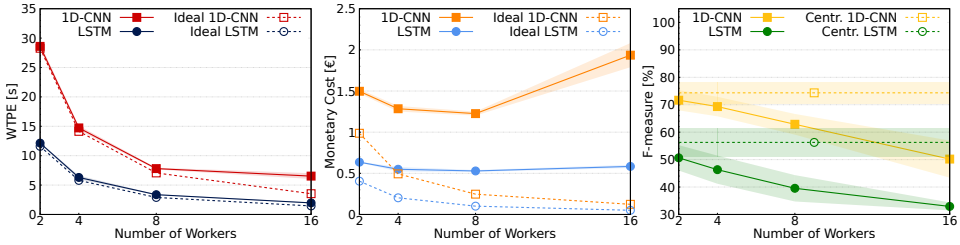
Cost vs. Number of Workers (N). Figs. 6.2b, 6.2e, and 6.2h show the impact of the number of workers ($N \in \{2, 4, 8, 16\}$) on the training cost of the two considered DL architectures in line with the pay-as-you-go billing model enforced, when addressing both binary and multi-class mobile TC tasks (F is again set to one update for epoch). To stress the overhead cost incurred by BD-enabled DL architectures, we also report (for each architecture) the *ideal-cost curve* corresponding to $(\rho\, N + \rho_M) \cdot (T_1/N)$, namely the cost required to train the DL architecture in the ideal case the BD framework guarantees perfect parallelization, being a *lower-bound* on the achievable cost—with $\rho_M = \rho$ in our case (see §6.3.1). While the *hourly cost* for cloud system deployment linearly increases with N (namely $\rho\,(N + 1)$), the resulting total cost C for completing the training phase is



(a) (FB/FBM) WTPE. (b) (FB/FBM) Monetary Cost. (c) (FB-FBM) F-measure.



(d) (Android) WTPE. (e) (Android) Monetary Cost. (f) (Android) F-measure.



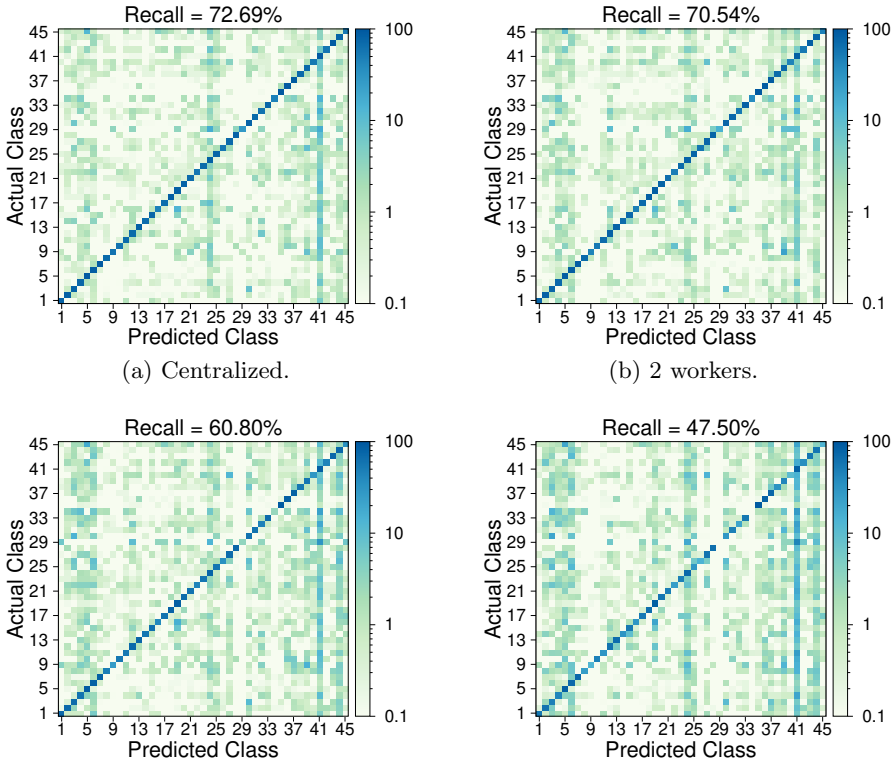
(g) (iOS) WTPE. (h) (iOS) Monetary Cost. (i) (iOS) F-measure.

Figure 6.2: Impact of the number of workers on WTPE, Monetary cost, and F-measure for FB/FBM (a, b, and c), Android (d, e, and f), and iOS (g, h, and i) datasets. Both 1D-CNN and LSTM architectures are considered. Average on 10-folds with $\pm 3\sigma$ confidence bands are shown.

also proportional to the required time T. As the training time may deviate from its ideal value as shown in the previous analysis for higher values of N (*e.g.*, only negligible benefits are achieved moving N from 8 to 16 when using 1D-CNN for Android and iOS), similarly, the resulting monetary cost may increase as the decreased training time does not always match a balanced gain in terms of hourly cost. Accordingly, while the deployment cost for LSTM and 1D-CNN for the FB/FBM dataset almost saturates for larger values of N, this is not the case for the multi-class ones. Indeed, the LSTM (for Android) and the 1D-CNN (for all multi-class TC tasks) show a clear increasing trend of monetary cost with N. In the latter cases (see Figs. 6.2e and 6.2h), deploying a larger number of workers ($N = 16$) leads to significantly higher costs (referring to the centralized deployment) for both Android (+24.9%) and iOS (+54.2%) with respect to the case $N = 8$, while the benefit in terms of reduced training time is negligible (-3.1% and -2.3%, respectively).

Classification Effectiveness vs. Number of Workers (N). Figs. 6.2c, 6.2f, and 6.2i report the effectiveness of the two DL architectures accomplishing binary and multi-class mobile TC tasks, when deployed on clusters where N ranges from 2 to 16. Experimental results witness (solid lines) how the degree of parallelization hinders the classification performance achieved, with F-measure values significantly decreasing as N increases. Accordingly, the worst classification performance is observed when relying on a 16-node cluster, namely -53.4% (resp. -41.5%) compared to a *centralized solution* when addressing FB/FBM (resp. iOS) classification via 1D-CNN (resp. LSTM). Furthermore, although in the Android setup F-measure shows the least deterioration for $N = 16$ nodes (*i.e.* -23.3% and -31.4% for LSTM and 1D-CNN, respectively), we can notice a significant variabil-

ity of performance—as denoted by the higher standard deviation—due to the harder TC task, also characterized by considerable class-imbalance (see Tab. 2.4), hardening the balanced splitting of the dataset among the workers. On the other hand, classification performance obtained by 2-node deployments are closer to those attainable by centralized DL implementations (dashed lines).



(a) Centralized.

(b) 2 workers.

(c) 8 workers.

(d) 16 workers.

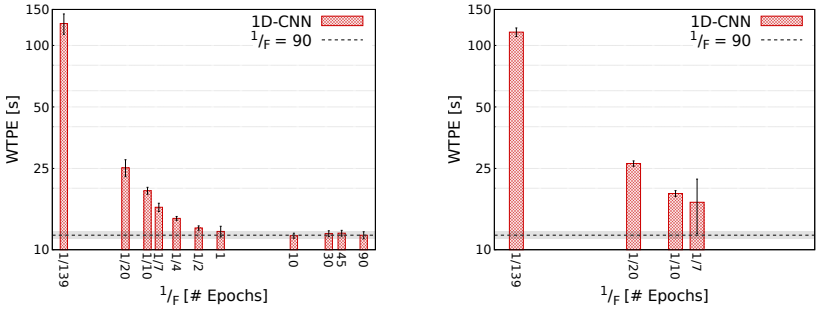
Figure 6.3: Confusion matrices of 1D-CNN on iOS dataset for centralized case (a) and BD-solutions with $N \in \{2, 4, 16\}$ workers (b-c-d) ([%] in log scale).

To deepen the above investigation, we show in Fig. 6.3 the confusion ma-

trices pertaining to the (best performing) 1D-CNN on iOS dataset—being more interesting than the binary TC task and showing severer deterioration with increasing N with respect to Android—by investigating the error patterns for $N \in \{2, 8, 16\}$ in comparison to the centralized case. The confusion matrices show a general degradation with growing N, with some apps *not recognized in most of the cases*, as also confirmed by the corresponding recall, namely 47.50% for $N = 16$ against 72.69% of the centralized deployment.

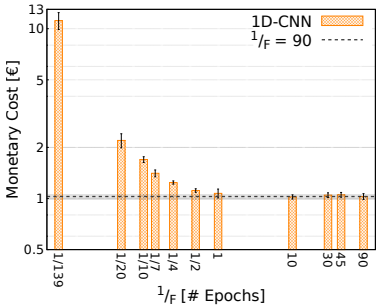
Such results witness how the adoption of DL deployments leveraging the power of BD frameworks may generate significant performance loss: though current solutions provide ready-to-use implementations with interfaces similar to (if not matching) the centralized counterparts, DL training stage is *not naturally parallelizable*, thus resulting in worse classification results due to reduced training accuracy collectively provided by workers when operating on smaller dataset portions.

Impact of worker update frequency (F). In Figs. 6.4a, 6.4c, and 6.4e, we evaluate the three considered dimensions versus worker update period $1/F$ (reported in terms of either number or fraction of epochs), with a range $\frac{1}{F} \in [1/139, 90]$ epochs (*i.e.* from one update every mini-batch to one update during the whole training phase). For budget constraints, the analysis focuses on the best performing BD-enabled DL architecture (*i.e.* 1D-CNN) trained and tested on the binary FB/FBM dataset with $N = 4$ workers. For both WTPE and cost analyses (Figs. 6.4a and 6.4c), we consider as the *lower-bound counterparts* the values obtained considering the loosest coupling between the workers and the master ($\frac{1}{F} = 90$), while for the F-measure (Fig. 6.4e) the *upper-bound* value of the centralized case. As expected, both WTPE and cost (Figs. 6.4a–6.4c) increase with F. Interestingly, a steep re-

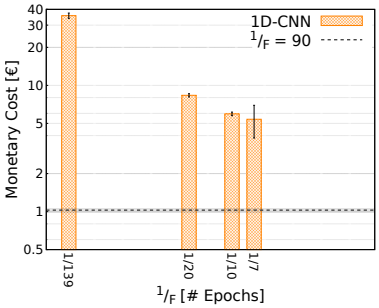


(a) WTPE (DS4v2).

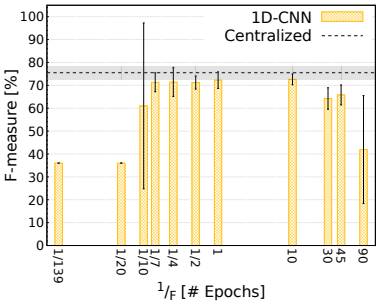
(b) WTPE (D32sv3).



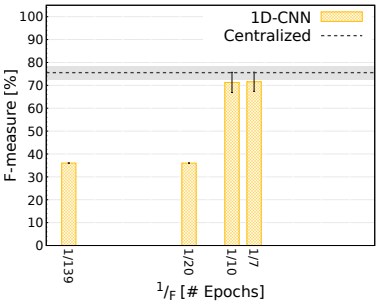
(c) Monetary Cost (DS4v2).



(d) Monetary Cost (D32sv3).



(e) F-measure (DS4v2).



(f) F-measure (D32sv3).

Figure 6.4: Impact of the number of epochs (reciprocal of worker update frequency) on WTPE (a and b in log-scale), Monetary cost (c and d in log-scale), and F-measure (e and f) for the FB/FBM dataset and 1D-CNN architecture using DS4v2 (a, c, and e) and D32sv3 (b, d, and f) nodes (the latter for $\frac{1}{F} \leq \frac{1}{7}$). Average on 10-folds with $\pm 3\sigma$ confidence intervals are shown.

duction is evident when passing from one update every single mini-batch to one every 7 mini-batches (*i.e.* from $\frac{1}{F} = \frac{1}{139}$ to $\frac{1}{F} = \frac{1}{20}$ epoch) with a -80.3% decrease. On the other hand, when the update period goes from $\frac{1}{F} = \frac{1}{20}$ to $\frac{1}{F} = 90$ the decrease is only -53.2%.

Finally, Fig. 6.4e shows the classification effectiveness in terms of F-measure. The best performance is obtained with $\frac{1}{4} \leq \frac{1}{F} \leq 10$ with a significant degradation for $\frac{1}{F} \leq \frac{1}{10}$ and $\frac{1}{F} \geq 30$. Whilst worse performance is expected when the exchange of updates is less frequent (*i.e.* $\frac{1}{F} \geq 30$, right side of Fig. 6.4e), this phenomenon is unexpected in the presence of tight coupling (*i.e.* $\frac{1}{F} \leq \frac{1}{10}$, left side of Fig. 6.4e).

To shed light on this evidence, we have performed additional focused experiments with better-performing worker/master nodes (`D32sv3`) for $\frac{1}{F} \leq \frac{1}{7}$ reported in Figs. 6.4b, 6.4d, and 6.4f. Results highlight that WTPE and cost have similar trends to those depicted in Figs. 6.4a and 6.4c, with higher costs due to the more expensive node configuration adopted (see §6.3.1). Conversely, in this case the F-measure obtained with $\frac{1}{F} = \frac{1}{7}$ and $\frac{1}{F} = \frac{1}{10}$ is comparable with the best-performing case, thus not showing any performance decrease due to the tight coupling. Nonetheless, the same performance trend of Fig. 6.4e is observed for $\frac{1}{F} \leq \frac{1}{20}$. This result suggests that a *computational bottleneck* exists at the master, hindering the correct collection of the updates from the workers, hence resulting in a worse-performing DL model.

Chapter 7

Conclusions

In the last years network operators have experienced tremendous growth of network traffic, mostly generated by mobile devices, providing users an immediate and ubiquitous way to work, communicate, access content and services, etc. over the Internet. Recent reports [5, 6] forecast that the mobile subscriptions will be more than seven billions by 2024, with a compound annual growth rate ten times higher than that of subscriptions of broadband fixed connections. To face this unique challenge, several network players employ increasingly sophisticated network monitoring and management systems. These tools (*e.g.*, security and quality-of-service enforcement devices, network monitors, firewalls, etc.) require knowledge about network traffic and, more specifically, about the mobile apps generating it. Thus, mobile traffic analysis and particularly mobile traffic classification play a role of paramount importance in the management of current and future networks. However, if on the one hand, mobile TC provides valuable profiling information that can be used by several stakeholders, such as advertisers, healthcare and insurance companies, on the other hand, it reveals privacy downsides related, for example, to the identification of health and dating

apps or to its application in combination with the enforcement of bring-your-own-device policy. Concurrently, the broad adoption of encrypted protocols (TLS) and dynamic ports, along with the spread of apps communicating via HTTP(S) and privacy-preserving services (*e.g.*, ATs), blocks the road to accurate TC, defeating traditional deep packet inspection and port-based techniques. This paves the way to advanced ML and DL techniques that, starting from their solid know-how established in several fields, can help facing the new challenges of network-level traffic analysis and classification emerging in the mobile and encrypted scenario.

Fueled by these motivations, the Thesis has proposed *an extensive set of novel methodologies for mobile and encrypted TC* that advances the state-of-the-art from different points of view, considering both advanced ML-based approaches and innovative DL architectures, also dissecting the trade-offs of their deployment on BD cloud platforms. Moreover, the present contributions allow to answer as many different emerging *research questions* on mobile and encrypted TC (cf. §1.4.1).[1]

Contextually, the present dissertation has also faced a challenge specifically important in (data-driven) research on TC, being affected by the lack of timely and reliably-labeled public datasets generated by real human users. This issue is further worsened for mobile traffic, where the possibility of sharing significant and up-to-date datasets is hindered by both the highest privacy concerns and fast-paced evolution of traffic mix. In view of these considerations, the *MIRAGE* architecture for app-traffic capture and high accurate ground-truth creation has been designed, implemented, and exploited for the construction of the *MIRAGE-2019* dataset. The latter encompasses the traffic generated by more than 280 experimenters using 40

[1]For convenience, the research questions are reported herein as footnotes.

mobile apps via 3 devices and provides both per-(bi)flow features and per-packet data. The flexibility of the MIRAGE-2019 dataset has been proven by performing mobile traffic characterization and modeling tasks at various granularities. Also, a self-contained portion of MIRAGE-2019 (*i.e.* comprising the traffic of Facebook and Facebook Messenger apps), along with two multi-class datasets (*i.e.* consisting of Android and iOS apps' traffic) and the public Anon17 one, has been leveraged to define a *common benchmark* for the set of TC methodologies. This has allowed to operate in a more realistic and heterogeneous scenario with respect to related works, either considering iOS or Android traffic usually generated using automated bot-monkeys. Additionally, thanks to this setup, the anonymization level provided by the most common ATs could be also investigated. Furthermore, to promote the replicability of the analyses and the extension to multiple use cases, MIRAGE-2019 has been publicly released as an open dataset.

The first envisioned enhancement has resulted in the composition of traditional ML classifiers in advanced structures, able to better operate in the dynamic and challenging mobile and encrypted context.

Firstly, the TC of mobile apps has been tackled by proposing a *MCS encompassing hard and soft classifier-fusion techniques*. For the latter, several soft-combination methods belonging to different philosophies have been explored: CC trainable/non-trainable and CI combiners. Specifically, the considered MCS employed a pool of 7 state-of-the-art classifiers specific or suitable for mobile traffic and its performance has been evaluated on the common benchmark of human-generated mobile apps' traffic. The results have shown a performance gain of the MCS over the best base classifier, up to +9.5% macro recall in the case of Android traffic. Such improvement is quite general over the different apps considered, given the homogeneously-reduced error-patterns observed. Nonetheless, the modularity of the consid-

ered MCS allows its virtual application to other suitably-devised classifiers and the appropriate selection of an optimized subset to obtain further performance enhancement with possibly lower complexity (cf. RQ_1[2]).

Secondly, an *HC approach has been conceived and applied to the encrypted TC of ATs*, namely Tor, I2P, and JonDonym. This hierarchical framework capitalizes the class-structure of encrypted ATs' traffic by exploiting the "divide-et-impera" principle and, beyond classification performance gains, carries several advantages by design, in terms of modularity, training efficiency, distributed deployment, and "tunable view" of classification outputs (cf. RQ_2[3]). In detail, the proposed framework has been designed with varying constraints, resulting in implementations with different degrees of complexity in terms of classifiers, features, and reject option. The experimental analysis—carried on the Anon17 public dataset—has allowed reasoning on which degree anonymous traffic can be told apart, considering different granularities such as the anonymity network adopted, the traffic type tunneled in the network, and the application category generating such traffic. Moving from naïve to fine-grain optimized implementations for the hierarchical classifier, the results at different granularities have highlighted how HC guarantees interesting performance gains, especially at the finest (application) level, and confines misclassifications in the same anonymous network and even in the same traffic type. They have also witnessed the appeal and effectiveness of the HC framework in real environments and provided insights in identifying performance bottlenecks which lie on a very limited set of nodes in the hierarchy (*e.g.*, the applications running within

[2] RQ_1: to what extent is it possible to improve the classification performance of mobile apps taking the best from each state-of-the-art ML classifier via advanced combining techniques?

[3] RQ_2: is it possible to capitalize the inherent hierarchical class-structure of (encrypted) traffic to achieve various advantages in TC at increasingly finer granularity?

I2P are the hardest to told apart).

Since the successful adoption of ML traffic classifiers resorts to the design of meaningful handcrafted features, thanks to domain experts, such process is impractical when facing the fast-paced mobile traffic evolution as well as its peculiarities (*e.g.*, large number of similar apps, embedding of common third-party services, periodical OS and app updates, etc.), because it can be neither automated nor crowdsourced to non-experts due to the high specialization required. Therefore, the present dissertation has envisioned the application of DL as the disruptive breakthrough toward the automatic design of accurate inference systems able to capture complex dependencies among mobile-traffic data, thus limiting human expert intervention. The lack of a comprehensive and principled approach to DL-based classifiers applied to traffic has been one of the main motivation of this analysis.

Specifically, the Thesis has presented the development of a *systematic framework for the design, evaluation, and comparison of traffic classifiers via DL*, constituting a vital groundwork for sound advances on the general mobile and encrypted TC topic. Precisely, this investigation has enabled the surfacing of a list of guidelines and sparks, and highlighted caveats of traffic analysis domain, so as to avoid pitfalls in the design and evaluation of DL-based mobile traffic classifiers and be the springboard of real-world implementations (cf. RQ_3[4]). It has been found that the presence of different DL architectures raises the need of a performance evaluation workbench, based on well-defined metrics and a common benchmark. Also, given the tricky nature of mobile traffic, its segmentation is often implicit or overlooked, while unfocused input-data selection causes biased inputs being fed to DL classifiers, jeopardizing the validity of (inflated) results. Associated

[4] RQ_3: how can mobile and encrypted TC benefit from the domain-aware application of DL to capitalize the heterogeneous traffic nature and avoid biased outcomes?

with this lesson learned, the challenge of carefully analyzing and selecting the input of DL algorithms has been tackled. Unluckily, an elaborated input selection process contrasts the DL promise of the reduced need of domain expertise, with some studies even preliminarily extracting features from data, instead of leveraging DL for that. In the case of DL-based classifiers, this issue is worsened by the black-box nature of most algorithms, as the performance impact of specific inputs is barely or not-at-all predictable. Hence, striking the right balance between naïve application and expertise-driven effort constitutes a still open challenge. Moreover, though DL architectures relieve the designer from the feature design issue, they come with many hyper-parameters to be tuned (*e.g.*, the optimizer, the number of layers/hidden nodes, the regularizers). To explore the performance gain brought by fine-grained design, this further process can be as complex and resource-demanding as feature design and it is substantially overlooked in most related works. On the plus side, differently from feature design of ML solutions this process can be automated, as it is less domain-driven.

Starting from the design milestones emerged from the proposed framework and observing that the choice of a DL traffic classifier seldom well matches with the nature of input data, the heterogeneous information available from traffic has been capitalized through the definition of a *general multi-modal DL architecture* (named *MIMETIC*), the proposal of a sophisticated training procedure, and the implementation of a specific MIMETIC instance. The experimental evaluation has demonstrated that the latter implementation outperforms both ML- and single-modal DL-based baselines, with up to +8.66% F-measure improvement over the best baseline (*i.e.* 82.99% on the iOS dataset), while having a RTPE $> 3.5\times$ lower than its "main single-modal DL competitor".

Finally, *mobile TC via DL architectures supported by BD solutions* has

been analyzed by realizing an actual deployment on the Microsoft Azure public-cloud BD platform. A systematic evaluation has pursued along three intertwined dimensions, namely training completion time, monetary costs, and classification performance, employing the mobile common benchmark. Accordingly, interesting outcomes and useful guidelines have been produced for harvesting the benefits deriving from the joint adoption of DL and BD, with specific focus on mobile TC. In detail, the above outcomes have highlighted the dependence of BD-enabled DL-based mobile traffic classifiers, in a non-trivial way, on the degree of parallelization and on the communication frequency of the BD architecture supporting the training phase of DL-based traffic classifiers. Although the adoption of the BD framework to support DL architectures significantly reduces the overall training time, especially in the case of high parallelization, its non-transparent nature has a direct implication on DL classification performance. Indeed, the joint use of data parallelism and federated learning provides a final trained DL architecture representing a sub-optimal solution to the TC task, not reaching the performance of a centralized solution, that takes longer times, but works on the training set as a whole, with more marked effects in the higher parallelization cases (cf. RQ_4[5]). Such performance gap significantly depends also on the worker update frequency, and TC "centralized" performance may be approached only through higher frequency values. Sadly, this inherent trade-off leads to higher computational overhead for the master (viz. more powerful hardware required) and impacts on both time and cost performance. This precludes a wallclock time cut proportionally to the number of workers, which reflects on the cost unsuitability, highlighted by a cost-optimal number of workers.

[5] RQ_4: which are the implications on TC when adopting BD solutions to deal with the demanding training phase of DL-based traffic classifiers?

Thanks to the set of proposed methodologies for mobile and encrypted TC, various possible directions to improve the state-of-the-art approaches have been devised, critically studied, and experimentally evaluated. In detail, this Thesis has envisioned both enhancements of standard ML classifiers and the systematic adoption of the innovative DL paradigm, also declined in advanced multi-modal fashion. Therefore, the analyzed solutions represent increasingly-advanced aspects of a composite comprehensive TC framework explicitly devised to deal with the unique traits of encrypted and mobile traffic, bringing out a number of lessons learned and open issues that may be useful also to further encourage advancements in this cutting-edge topic.

In this regard, possible suggestions for future research could be the proposal of more sophisticated compositions (based on both "horizontal" and "vertical" structures) of classifiers, operating with different traffic objects and with applications arranged in "multi-view" structures, or fed with a specifically-optimized set of features (*e.g.*, selected by means of information-theoretic measures and whose stability, with respect to the dynamic nature of mobile traffic, needs to be carefully evaluated). For example, an OS-agnostic classification concerning the possibility to distinguish not only the specific app, but also the OS it belongs, represents an interesting further avenue. On the other hand, additional analyses on the inner structure of DL networks can be also conducted along the lines of *explainable AI* [180], a recent field of study that has yet to see application to TC. Complementary to the latter, further performance gain is foreseen via the exploitation of massive unsupervised data (granted by BD solutions) for improved learning, along with the use of pre-trained architectures (*i.e.* transfer learning) and sophisticated learning layers (*e.g.*, inception and residual connections). Additionally, although some efforts have been made from a system viewpoint, design and real-world implementations (*e.g.*, in open-source tools such as

TIE [40]) of accurate multi-modal/multi-task DL architectures are still unexplored. Finally, advanced deployment could be envisioned by prototyping of BD-enabled DL architectures for TC able to exploit both model and data parallelism or leveraging stream-based learning implementations [174].

Appendix A

Ethical consideration in MIRAGE-2019 collection

In this Appendix, we discuss the ethical considerations underlining the collection of the MIRAGE-2019 dataset described in Chapter 2. Based on both the presented design choices and the experimental setup, the capture process (cf. §2.2) and the collected dataset (cf. §2.3) do not imply any ethical concern [108]. We remark that experimenters involved in the acquisition phase have been beforehand informed and warned about the objectives of their activities (*e.g.* network traffic analysis) and the possible public release of the corresponding traces for research purposes, even in a complete form (although not being the case of MIRAGE-2019). Additionally, each experimenter received a thirty-minutes training phase (with the help of a written document listing all the instructions) regarding the capture technologies employed and the related working principles.

Moreover, the employed mobile devices were provided and used within the ARCLAB laboratory at the University of Napoli "Federico II". The logged source IP addresses belong to the *private* IPv4 space, with no privacy implications. Equally important, purposedly-created app accounts were em-

ployed for the capture sessions (cf. §2.2.1 and §2.3). The GT building phase as described in Sec. 2.2.2 is automated and does not contribute any additional experimenter-related information. Hence, no personal information of the experimenters was involved at any time from the capture phase to the dataset creation.

Finally, experimenters have been also involved in the analysis and inspection of traces. Specifically, collected traces were made available to experimenters that captured them for educational purposes as well as to verify the non-sensitive nature of their contents.

Acknowledgments

First of all, I wish to express my sincere gratitude to my advisor Prof. Antonio Pescapé for his guidance and inspiration during these years, first as a student, then as a researcher. A heartfelt thanks go also to my labmates (in rigorous alphabetical order) Domenico, Fabio, Giampaolo, Giuseppe, and Valerio. With their different approaches and points of view, they have been and are a priceless source of inspiration. Every discussion and suggestion has helped me to face challenges and also glitches in a scientific and rigorous manner. Besides, all of them are not only colleagues but also friends. I would like to state my appreciation to Prof. Özgü Alay and Prof. Narseo Vallina-Rodriguez, my internship advisors at the SIMULA Research Laboratory and IMDEA Networks Institute, respectively. They gave me different perspectives on research, playing an important role in my professional growth. I am also grateful to Prof. Marilia Curado (University of Coimbra, Portugal) and Prof. Benoit Donnet (Université de Liège, Belgium) for their valuable comments on the draft version of my Ph.D. Thesis, which allowed me to improve it significantly. My thanks and love go to my parents Gianna and Gianni, my sister Morena, my brother Flavio, and the rest of my big family for their unconditional and constant support. My gratitude goes also to my friends being able to make me smile, despite everything. A special thanks go to my beloved girlfriend Rosanna, for her love, patience, and

also pragmatism. She has been my leading light in many circumstances. This journey has been lighter since I met her. Finally, my constant thought is for my Angel. I miss you more than I can express with the words.

Bibliography

[1] We Are Social. Digital 2019 report. https://wearesocial.com/global-digital-report-2019/. Online; accessed Dec. 2019.

[2] J. Clement. Global digital population as of July 2019. https://www.statista.com/statistics/617136/digital-population-worldwide/. Online; accessed Dec. 2019.

[3] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

[4] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, 2009.

[5] Fredrik Jejdling et al. Ericsson mobility report. *Ericsson AB, Business Area Networks, Stockholm, Sweden, Tech. Rep. EAB-19*, 3442, 2019.

[6] Sandvine. The 2019 mobile internet phenomena report. https://www.sandvine.com/hubfs/downloads/phenomena/2019-mobile-phenomena-report.pdf. Online; accessed Dec. 2019.

[7] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. "won't somebody think of the children?" examining coppa compliance at scale. *Proceedings on Privacy Enhancing Technologies*, 2018(3):63–83, 2018.

[8] Jody Farnden, Ben Martini, and Kim-Kwang Raymond Choo. Privacy risks in mobile dating apps. *arXiv preprint arXiv:1505.02906*, 2015.

[9] A. Y. Nikravesh, S. A. Ajila, C. Lung, and W. Ding. Mobile network traffic prediction using mlp, mlpwd, and svm. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 402–409, 2016.

[10] C. Zhang and P. Patras. Long-term mobile traffic forecasting using deep spatio-temporal neural networks. In *18th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc)*, pages 231–240. ACM, 2018.

[11] Laisen Nie, Dingde Jiang, Shui Yu, and Houbing Song. Network traffic prediction based on deep belief network in wireless mesh backbone networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–5, 2017.

[12] M. S. Parwez, D. B. Rawat, and M. Garuba. Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, 13(4):2058–2065, 2017.

[13] P. I. Radoglou-Grammatikis and P. G. Sarigiannidis. Flow anomaly based intrusion detection system for android mobile devices. In *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4, 2017.

[14] G. Marín, P. Casas, and G. Capdehourat. Rawpower: Deep learning based anomaly detection from raw network traffic measurements. In *ACM SIGCOMM Conference on Posters and Demos*, pages 75–77, 2018.

[15] V. L. L. Thing. Ieee 802.11 network anomaly detection and attack classification: A deep learning approach. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2017.

[16] Ragav Sridharan, Rajib Ranjan Maiti, and Nils Ole Tippenhauer. Wadac: Privacy-preserving anomaly detection and attack classification on wireless traffic. In *Proceedings of the 11th ACM Conference*

on Security & Privacy in Wireless and Mobile Networks, WiSec '18, pages 51–62, New York, NY, USA, 2018. ACM.

[17] Abebe Diro and Naveen Chilamkurti. Leveraging lstm networks for attack detection in fog-to-things communications. *IEEE Communications Magazine*, 56(9):124–130, 2018.

[18] Anshul Arora and Sateesh K. Peddoju. Minimizing network traffic features for android mobile malware detection. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, pages 32:1–32:10, New York, NY, USA, 2017. ACM.

[19] Yun-Chun Chen, Yu-Jhe Li, Aragorn Tseng, and Tsungnan Lin. Deep learning for malicious flow detection. In *IEEE 28th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7, 2017.

[20] Zhenxiang Chen, Qiben Yan, Hongbo Han, Shanshan Wang, Lizhi Peng, Lin Wang, and Bo Yang. Machine learning based mobile malware detection using highly imbalanced network traffic. *Information Sciences*, 433:346–364, 2018.

[21] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng. Malware traffic classification using convolutional neural network for representation learning. In *IEEE International Conference on Information Networking (ICOIN)*, pages 712–717, 2017.

[22] A. H. Lashkari, A. F. A.Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309, 2017.

[23] He Huang, Haojiang Deng, Jun Chen, Luchao Han, and Wei Wang. Automatic multi-task learning system for abnormal network traffic detection. *International Journal of Emerging Technologies in Learning*, 13(04):4–20, 2018.

[24] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting

attacks on android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, pages 49–60, New York, NY, USA, 2016. ACM.

[25] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[26] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM Conference on Computer and Communications Security (SIGSAC)*, pages 1928–1943, 2018.

[27] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, 2018.

[28] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys Tutorials*, 21(3):2224–2287, 2019.

[29] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing traffic classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer, 2013.

[30] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *IEEE network*, 26(1):35–40, 2012.

[31] Arthur Callado, Carlos Kamienski, Géza Szabó, Balázs Péter Gero, Judith Kelner, Stênio Fernandes, and Djamel Sadok. A survey on internet traffic identification. *IEEE Communications Surveys & Tutorials*, 11(3):37–52, 2009.

[32] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[33] Junghun Park, Hsiao-Rong Tyan, and C-C Jay Kuo. Ga-based internet traffic classification technique for qos provisioning. In *2006 International Conference on Intelligent Information Hiding and Multimedia*, pages 251–254. IEEE, 2006.

[34] R. Pries, F. Wamser, D. Staehle, K. Heck, and P. Tran-Gia. Traffic measurement and analysis of a broadband wireless internet access. In *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, pages 1–5, 2009.

[35] Z. Zhou, L. Yao, J. Li, B. Hu, C. Wang, and Z. Wang. Classification of botnet families based on features self-learning under network traffic censorship. In *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–7, 2018.

[36] Z. Li, X. Wang, N. Huang, M. A. Kaafar, Z. Li, J. Zhou, G. Xie, and P. Steenkiste. An empirical analysis of a large-scale mobile cloud storage service. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 287–301, New York, NY, USA, 2016. ACM.

[37] John P. Rula, Fabián E. Bustamante, and Moritz Steiner. Cell spotting: Studying the role of cellular networks in the internet. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, pages 191–204, New York, NY, USA, 2017. ACM.

[38] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[39] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *ACM CoNEXT conference*, page 6, 2006.

[40] Walter De Donato, Antonio Pescapé, and Alberto Dainotti. Traffic identification engine: an open platform for traffic classification. *IEEE Network*, 28(2):56–64, 2014.

[41] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour.

In *ACM 6th conference on Security and privacy in wireless and mobile networks (WISEC)*, pages 7–12, 2013.

[42] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454, 2016.

[43] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2018.

[44] K. Shahbar and A. N. Zincir-Heywood. Packet momentum for identification of anonymity networks. *Journal of Cyber Security and Mobility*, 6(1):27–56, 2017.

[45] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM computer communication review*, volume 35, pages 229–240. ACM, 2005.

[46] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Byte me: A case for byte accuracy in traffic classification. In *Proceedings of the 3rd Annual ACM Workshop on Mining Network Data*, MineNet '07, pages 35–38, New York, NY, USA, 2007. ACM.

[47] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. Traffic analysis with deep learning. *arXiv*, 2017.

[48] C. Shen and L. Huang. On detection accuracy of l7-filter and opendpi. In *2012 Third International Conference on Networking and Distributed Computing*, pages 119–123, 2012.

[49] Rafael Antonello, Stênio Fernandes, Carlos Kamienski, Djamel Sadok, Judith Kelner, István Gódor, Géza Szabó, and Tord Westholm. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. *J. Netw. Comput. Appl.*, 35(6):1863–1878, 2012.

[50] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622, 2014.

[51] Robert Todd Graham Collins. The privacy implications of deep packet inspection technology: Why the next wave in online advertising shouldn't rock the self-regulatory boat. *Ga. L. Rev.*, 44:545, 2009.

[52] Alissa Cooper. Doing the dpi dance: Assessing the privacy impact of deep packet inspection. *Privacy in America. Interdisciplinary perspectives*, pages 139–165, 2011.

[53] G. Aceto, A. Dainotti, W. de Donato, and A. Pescape. Portload: Taking the best of two worlds in traffic classification. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–5, 2010.

[54] Sandvine. Global Internet Phenomena Spotlight: Encrypted Internet Traffic., 2016.

[55] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill. Studying TLS usage in Android apps. In *13th ACM CoNEXT*, pages 350–362, 2017.

[56] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *International Workshop on Passive and Active Network Measurement*, pages 41–54. Springer, 2005.

[57] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.

[58] Donald Michie. "memo" functions and machine learning. *Nature*, 218(5136):19, 1968.

[59] Zhongzhi Shi. *Principles of machine learning*. International Academic Publishers, 1992.

[60] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2016.

[61] Amjad Hajjar, Jawad Khalife, and Jesús Díaz-Verdejo. Network traffic application identification based on message size analysis. *Journal of Network and Computer Applications*, 58:130–143, 2015.

[62] A. Dainotti, F. Gargiulo, L. I. Kuncheva, A. Pescapé, and C. Sansone. Identification of traffic flows hiding behind TCP port 80. In *2010 IEEE International Conference on Communications*, pages 1–6, 2010.

[63] Chun-Nan Lu, Chun-Ying Huang, Ying-Dar Lin, and Yuan-Cheng Lai. High performance traffic classification based on message size sequence and distribution. *Journal of Network and Computer Applications*, 76:60–74, 2016.

[64] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *ACM 13th conference on Computer and communications security (CCS)*, pages 255–263, 2006.

[65] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier. In *ACM workshop on Cloud computing security*, pages 31–42, 2009.

[66] Ying-Dar Lin, Chun-Nan Lu, Yuan-Cheng Lai, Wei-Hao Peng, and Po-Ching Lin. Application classification using packet size distribution and port association. *Journal of Network and Computer Applications*, 32(5):1023–1030, 2009.

[67] Aarti Singh, Robert Nowak, and Jerry Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in neural information processing systems*, pages 1513–1520, 2009.

[68] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[69] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. Studying tls usage in android apps. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 350–362, New York, NY, USA, 2017. ACM.

[70] P. Syverson, R. Dingledine, and N. Mathewson. Tor: the second generation onion router. In *USENIX 13th Security Symposium (SSYM)*, 2004.

[71] Tim Majchrzak and Tor-Morten Grønli. Comprehensive analysis of innovative cross-platform app development frameworks. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[72] G. Szabo, I. Szabo, and D. Orincsay. Accurate traffic classification. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–8, 2007.

[73] Haitao He, Chunhui Che, Feiteng Ma, Jun Zhang, and Xiaonan Luo. Traffic classification using en-semble learning and co-training. In *Proceedings of the 8th conference on Applied informatics and communications*, pages 458–463. World Scientific and Engineering Academy and Society (WSEAS), 2008.

[74] Arthur Callado, Judith Kelner, Djamel Sadok, Carlos Alberto Kamienski, and Stênio Fernandes. Better network traffic identification through the independent combination of techniques. *Journal of Network and Computer Applications*, 33(4):433 – 446, 2010.

[75] J. Yu, H. Lee, Y. Im, M.-S. Kim, and D. Park. Real-time classification of internet application traffic using a hierarchical multi-class SVM. *KSII Transactions on Internet & Information Systems*, 4(5), 2010.

[76] Alberto Dainotti, Antonio Pescapé, and Carlo Sansone. Early classification of network traffic through multi-classification. In *International Workshop on Traffic Monitoring and Analysis*, pages 122–135. Springer, 2011.

[77] L. Grimaudo, M. Mellia, and E. Baralis. Hierarchical learning for fine grained internet traffic classification. In *8th International Wireless Communications and Mobile Computing Conference*, pages 463–468, 2012.

[78] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. Networkprofiler: towards automatic fingerprinting of Android apps. In *Proceedings of IEEE INFOCOM*, pages 809–817, 2013.

[79] Valerio D'Alessandro, Byungchul Park, Luigi Romano, Christof Fetzer, et al. Scalable network traffic classification using distributed support vector machines. In *IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 1008–1012. IEEE, 2015.

[80] Q. Wang, A. Yahyavi, B. Kemme, and W. He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *IEEE Conference on Communications and Network Security (CNS)*, pages 433–441, 2015.

[81] Zhanyi Wang. The applications of deep learning on traffic identification. In *Black Hat USA, Las Vegas*, 2015.

[82] J. h. Kim, S.-H. Yoon, and M.-S. Kim. Study on traffic classification taxonomy for multilateral and hierarchical traffic classification. In *14th IEEE Asia-Pacific Network Operations and Management Symposium*, pages 1–4, 2012.

[83] S.-H. Yoon, K.-S. Shim, S.-K. Lee, and M.-S. Kim. Framework for multi-level application traffic identification. In *17th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 424–427, 2015.

[84] H. F. Alan and J. Kaur. Can Android applications be identified using only TCP/IP headers of their launch time traffic? In *9th ACM*

Conference on Security & Privacy in Wireless and Mobile Networks (WiSec), pages 61–66, 2016.

[85] Wazen M Shbair, Thibault Cholez, Jérôme François, and Isabelle Chrisment. A multi-level framework to identify HTTPS services. In *IEEE/IFIP NOMS*, pages 240–248, 2016.

[86] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Trans. Inf. Forensics Security*, 11(1):114–125, 2016.

[87] Zhengwu Yuan and Chaozheng Wang. An improved network traffic classification algorithm based on Hadoop decision tree. In *IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, pages 53–56, 2016.

[88] Y. n. Dong, J. j. Zhao, and J. Jin. Novel feature selection and classification of internet video traffic based on a hierarchical scheme. *Computer Networks*, 119:102–111, 2017.

[89] Ding Li, Yuefei Zhu, and Wei Lin. Traffic identification of mobile apps based on variational autoencoder network. In *13th IEEE International Conference on Computational Intelligence and Security (CIS)*, pages 287–291, 2017.

[90] Z. Chen, K. He, J. Li, and Y. Geng. Seq2Img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *IEEE International Conference on Big Data (Big Data)*, pages 1271–1276, 2017.

[91] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access*, 5:18042–18050, 2017.

[92] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian. Deep packet: a novel approach for encrypted traffic classification using Deep Learning. *arXiv*, 2017.

[93] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. A deep learning based method for handling imbalanced problem in network traffic classification. In *ACM 8th International Symposium on Information and Communication Technology (SoICT)*, pages 333–339, 2017.

[94] W. Wang et al. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *IEEE Int. Conf. on Intelligence and Security Informatics (ISI)*, pages 43–48, 2017.

[95] Wenlong Ke, Yong Wang, Xiaochun Lei, and Bizhong Wei. Spark-based feature selection algorithm of network traffic classification. In *IEEE 13th International Conference on Computational Intelligence and Security (CIS)*, pages 140–144. IEEE, 2017.

[96] Xuejiao Chen, Jiahui Yu, Feng Ye, and Pan Wang. A hierarchical approach to encrypted data packet classification in smart home gateways. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/-CyberSciTech)*, pages 41–45. IEEE, 2018.

[97] Luong-Vy Le, Bao-Shuh Lin, and Sinh Do. Applying Big Data, Machine Learning, and SDN/NFV for 5G early-stage traffic classification and network QoS control. *Transactions on Networks and Communications*, 6(2):36, 2018.

[98] Hongtao Shi, Hongping Li, Dan Zhang, Chaqiu Cheng, and Xuanxuan Cao. An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification. *Computer Networks*, 2018.

[99] Chuangchuang Zhang, Xingwei Wang, Fuliang Li, Qiang He, and Min Huang. Deep learning–based network application classification for SDN. *Wiley Transactions on Emerging Telecommunications Technologies*, 13(04):4–20, 2018.

[100] P. Wang, F. Ye, X. Chen, and Y. Qian. Datanet: Deep learning based encrypted network traffic classification in SDN home gateway. *IEEE Access*, 2018.

[101] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia. Byte segment neural network for network traffic classification. In *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2018.

[102] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. FS-Net: A flow sequence network for encrypted traffic classification. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1171–1179, 2019.

[103] Haifeng Sun, Yunming Xiao, Jing Wang, Jingyu Wang, Qi Qi, Jianxin Liao, and Xiulei Liu. Common knowledge based and one-shot learning enabled multi-task traffic classification. *IEEE Access*, 7:39485–39495, 2019.

[104] Yi Zeng, Huaxi Gu, Wenting Wei, and Yantao Guo. $deep - full - range$: A deep learning based network encrypted traffic classification and intrusion detection framework. *IEEE Access*, 7:45182–45190, 2019.

[105] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet?(e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 429–440. IEEE, 2015.

[106] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *SIGMETRICS Perform. Eval. Rev.*, 33(1):50–60, 2005.

[107] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani. Characterization of encrypted and VPN traffic using time-related features. In *2nd Int. Conference on Information Systems Security and Privacy (ICISSP)*, pages 407–414, 2016.

[108] Vaibhav Bajpai, Anna Brunstrom, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis,

Matthias Wählisch, and Klaus Wehrle. The Dagstuhl beginners guide to reproducibility for experimental networking research. *ACM SIGCOMM Computer Communication Review*, 49(1):24–30, 2019.

[109] A. Dainotti, R. Holz, M. Kühlewind, A. Lutu, J. Sommers, and B. Trammell. Open collaborative hyperpapers: a call to action. *ACM SIGCOMM Computer Communication Review*, 49(1):31–33, 2019.

[110] A. H. Lashkari, G. Draper-Gil, M. Mamun, I. Saiful, and A. A. Ghorbani. Characterization of Tor traffic using time based features. In *3rd International Conference on Information System Security and Privacy*, pages 253–262, 2017.

[111] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk. A novel QUIC traffic classifier based on convolutional neural networks. In *IEEE International Conference on Global Communications (GlobeCom)*, pages 1–6, 2018.

[112] R. Wang, Z. Liu, Y. Cai, D. Tang, J. Yang, and Z. Yang. Benchmark data for mobile app traffic research. In *15th ACM EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, pages 402–411. ACM, 2018.

[113] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Trans. Mobile Comput.*, 2018.

[114] M. Lescisin and Q. H. Mahmoud. Dataset for web traffic security analysis. In *IEEE IECON'18*, pages 2700–2705. IEEE, 2018.

[115] R. Dubin, A. Dvir, O. Pele, and O. Hadar. I know what you saw last minute-encrypted HTTP adaptive video streaming title classification. *IEEE Trans. Inf. Forensics Security*, 12(12):3039–3049, 2017.

[116] T. Karagkioules, D. Tsilimantos, S. Valentin, F. Wamser, B. Zeidler, M. Seufert, F. Loh, and P. Tran-Gia. A public dataset for YouTube's mobile streaming client. In *IEEE/ACM Network Traffic Measurement and Analysis Conference (TMA) -*, pages 1–6. IEEE, 2018.

[117] S. C. Madanapalli, H. H. Gharakhieli, and V. Sivaraman. Inferring netflix user experience from broadband network measurement. In *IEEE/ACM Network Traffic Measurement and Analysis Conference (TMA)*, pages 569–574, 2019.

[118] Selenium. Seleniumhq. https://www.seleniumhq.org. Online; accessed Dec. 2019.

[119] tcpdump & libpcap. http://www.tcpdump.org/. Online; accessed Dec. 2019.

[120] Android developers. Android debug bridge (adb). https://developer.android.com/studio/command-line/adb.html. Online; accessed Dec. 2019.

[121] StatCounter. Global Stats - Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide. Online; accessed Dec. 2019.

[122] 42matters. Google play categories. https://42matters.com/docs/app-market-data/android/apps/google-play-categories. Online; accessed Dec. 2019.

[123] A. Dainotti, A. PescapÃ©, P. Salvo Rossi, F. Palmieri, and G. Ventre. Internet traffic modeling by means of hidden Markov models. *Computer Networks*, 52(14):2645 – 2662, 2008.

[124] The Invisible Internet Project (I2P). https://geti2p.net/en/. Online; accessed Dec. 2019.

[125] Project: AN.ON - Anonymity. http://anon.inf.tu-dresden.de/index_en.html, Online; accessed Dec. 2019.

[126] M. Hall, F. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[127] A. Tongaonkar. A look at the mobile app identification landscape. *IEEE Internet Computing*, 20(4):9–15, 2016.

[128] Alice Este, Francesco Gringoli, and Luca Salgarelli. On the stability of the information carried by traffic flow features at the packet level. *ACM SIGCOMM Computer Communication Review*, 39(3):13–18, 2009.

[129] Lizhi Peng, Bo Yang, Yuehui Chen, and Zhenxiang Chen. Effectiveness of statistical features for early stage internet traffic identification. *International Journal of Parallel Programming*, 44(1):181–197, 2016.

[130] L. I. Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[131] L. I. Kuncheva and J. J. Rodríguez. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, 38(2):259–275, 2014.

[132] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287. ACM, 2010.

[133] Taimur Bakhshi and Bogdan Ghita. On internet traffic classification: A two-phased machine learning approach. *Journal of Computer Networks and Communications*, 2016.

[134] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[135] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: prediction, inference and data mining*. Springer, 2009.

[136] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? *Computer networks*, 55(6):1326–1350, 2011.

[137] Y. S. Huang and C. Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94, 1995.

[138] D. M. J. Tax, R. P. W. Duin, and M. Van Breukelen. Comparison between product and mean classifier combination rules. In *Proceedings of the Workshop on Statistical Pattern Recognition, Prague, Czech*, 1997.

[139] Giorgio Fumera and Fabio Roli. Performance analysis and comparison of linear combiners for classifier fusion. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 47–64, 2002.

[140] Jon Atli Benediktsson, Johannes R Sveinsson, Okan K Ersoy, and Philip H Swain. Parallel consensual neural networks. *IEEE Transactions on Neural Networks*, 8(1):54–64, 1997.

[141] Qiong Liu and Zhen Liu. A comparison of improving multi-class imbalance for internet traffic classification. *Information Systems Frontiers*, 16(3):509–521, 2014.

[142] Lizhi Peng, Haibo Zhang, Yuehui Chen, and Bo Yang. Imbalanced traffic identification using an imbalanced data gravitation-based classification model. *Computer Communications*, 102:177–189, 2017.

[143] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[144] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[145] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks*, 83:381–421, 2015.

[146] J. Barker, P. Hannay, and P. Szewczyk. Using traffic analysis to identify the second generation onion router. In *9th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 72–78, 2011.

[147] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website fingerprinting at internet scale. In *Network and Distributed System Security Symposium*, 2016.

[148] G. He, M. Yang, J. Luo, and X. Gu. Inferring application type information from tor encrypted traffic. In *IEEE 2nd International Conference on Advanced Cloud and Big Data*, pages 220–227, 2014.

[149] M. AlSabah, K. Bauer, and I. Goldberg. Enhancing Tor's performance using real-time traffic classification. In *ACM Conference on Computer and Communications security*, pages 73–84, 2012.

[150] K. Shahbar and A. N. Zincir-Heywood. Benchmarking two techniques for Tor classification: Flow level and circuit level classification. In *IEEE Symposium on Computational Intelligence in Cyber Security*, pages 1–8, 2014.

[151] K. Shahbar and A. N. Zincir-Heywood. An analysis of Tor pluggable transports under adversarial conditions. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2017.

[152] K. Shahbar and A. N. Zincir-Heywood. Effects of shared bandwidth on anonymity of the I2P network users. In *IEEE Symposium on Security and Privacy, Workshop on Traffic Measurements for Cybersecurity (WTMC)*, pages 235–240, 2017.

[153] C. N. Silla and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[154] Zhihong Rao, Weina Niu, XiaoSong Zhang, and Hongwei Li. Tor anonymous traffic identification based on gravitational clustering. *Peer-to-Peer Networking and Applications*, pages 1–10, 2017.

[155] S. Burschka and B. Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *IEEE Symposium Series on Computational Intelligence*, pages 1–8, 2016.

[156] Tranalyzer2. http://tranalyzer.com, Online; accessed Dec. 2019.

[157] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[158] Alberto Dainotti, Antonio Pescapé, and Hyun-chul Kim. Traffic classification through joint distributions of packet-level statistics. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2011.

[159] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[160] K. Shahbar and A. N. Zincir-Heywood. Traffic flow analysis of Tor pluggable transports. In *IEEE 11th International Conference on Network and Service Management (CNSM)*, pages 178–181, 2015.

[161] J. R. Quinlan. *C4.5: programs for machine learning*. Elsevier, 2014.

[162] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[163] E. Hjelmvik and W. John. Breaking and improving protocol obfuscation. *Chalmers University of Technology, Tech. Rep*, 123751, 2010.

[164] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *28th International Conference on Machine Learning (ICML)*, pages 689–696, 2011.

[165] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard. Multimodal deep learning for robust RGB-D object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687. IEEE, 2015.

[166] D. Ramachandram and G. W. Taylor. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine*, 34(6):96–108, 2017.

[167] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark

datasets for intrusion detection. *Computers & Security*, 31(3):357 – 374, 2012.

[168] WITS: Waikato Internet Traffic Storage. https://wand.net.nz/wits/. Online; accessed Dec. 2019.

[169] C. Liu, Z. Cao, G. Xiong, G. Gou, S. Yiu, and L. He. Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2018.

[170] François Chollet et al. Keras. https://keras.io, 2015.

[171] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[172] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *34th International Conference on Machine Learning (ICML)*, ICML'17, pages 1321–1330, 2017.

[173] CERN IT-DB Joeri R. Hermans. Distributed keras: Distributed deep learning with apache spark and keras. https://github.com/JoeriHermans/dist-keras/, 2016.

[174] Pavol Mulinka and Pedro Casas. Stream-based machine learning for network security and anomaly detection. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 1–7. ACM, 2018.

[175] Eduardo Viegas, Altair Santin, Alysson Bessani, and Nuno Neves. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, pages 473–485, 2019.

[176] Baojun Zhou, Jie Li, Yusheng Ji, and Mohsen Guizani. Online internet traffic monitoring and DDoS attack detection using Big Data frameworks. In *14th IEEE International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1507–1512. IEEE, 2018.

[177] Valerio Persico, Antonio Pescapé, Antonio Picariello, and Giancarlo Sperlí. Benchmarking Big Data architectures for social networks data processing using public cloud platforms. *Future Generation Computer Systems*, 89:98–109, 2018.

[178] Mohammad Abu Alsheikh, Dusit Niyato, Shaowei Lin, Hwee-Pink Tan, and Zhu Han. Mobile Big Data analytics using Deep Learning and Apache Spark. *IEEE Network*, 30(3):22–29, 2016.

[179] Abebe Abeshu and Naveen Chilamkurti. Deep Learning: the frontier for distributed attack detection in Fog-to-Things computing. *IEEE Commun. Mag.*, 56(2):169–175, 2018.

[180] Hani Hagras. Toward human-understandable, explainable AI. *IEEE Computer*, 51(9):28–36, 2018.