POLITECNICO DI TORINO
Repository ISTITUZIONALE

Real-Time Classification of Real-Time Communications

Original
Real-Time Classification of Real-Time Communications / Perna, G., Markudova, D., Trevisan, M., Garza, P., Meo, M., Munafo, M., Carofiglio, G.. - In: IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. - ISSN 1932-4537. - ELETTRONICO. - 19:4(2022), pp. 4676-4690. [10.1109/TNSM.2022.3189628]

Availability:
This version is available at: 11583/2970070 since: 2025-02-08T13:45:49Z

Publisher:
IEEE

Published
DOI:10.1109/TNSM.2022.3189628

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

18 November 2026

Real-Time Classification of Real-Time Communications

Gianluca Perna†, Dena Markudova†, Martino Trevisan‡, Paolo Garza†,
Michela Meo†, Maurizio M. Munafò †, Giovanna Carofiglio ⋆,

†Politecnico di Torino, ‡University of Trieste, ⋆Cisco Systems Inc.,
first.last@polito.it, martino.trevisan@dia.units.it, gcarofig@cisco.com

***Abstract*—Real-time communication (RTC) applications have become largely popular in the last decade with the spread of broadband and mobile Internet access. Nowadays, these platforms are a fundamental means for connecting people and supporting businesses that increasingly rely on forms of remote work. In this context, it is of paramount importance to operate at the network level to ensure adequate Quality of Experience (QoE) for users, and appropriate traffic management policies are essential to prioritize RTC traffic. This in turn requires the network to be able to identify RTC streams and the type of content they carry.**

In this paper, we propose a machine learning-based application to classify media streams generated by RTC applications encapsulated in Secure Real-Time Protocol (SRTP) flows in real-time. Using carefully tuned features extracted from packet characteristics, we train models to classify streams into a variety of classes, including media type (audio/video), video quality, and redundant streams. We validate our approach using traffic from over 62 hours of multi-party meetings conducted using two popular RTC applications, namely Cisco Webex Teams and Jitsi Meet. We achieve an overall accuracy of 96% for Webex and 95% for Jitsi, using a lightweight decision tree model that makes decisions based solely on 1 second of real-time traffic. Our results show that models trained for a particular meeting software have difficulty when used with another one, although domain adaptation techniques facilitate the transfer of pre-trained models.

***Index Terms*—Real-Time Communication, RTP, Classification, Machine Learning.**

I. Introduction

In recent years, real-time communication (RTC) applications for video calls and virtual meetings have become a fundamental pillar of leisure and business. They help people communicate with each other and businesses save significant travel costs. Their value was especially proven during the months of self-isolation due to the COVID -19 pandemic, when online conferencing allowed many businesses to continue operations using remote working, mitigating the economic impact of the outbreak. This was largely possible due to the Internet being ubiquitous and the available bandwidth increasing [1]. After the first phase of IP telephony based on SIP and H.323, during the early 2000s, Skype opened the business for RTC applications, entering into competition with traditional telephony providers. At that time, most users were connected via cable modems, which offered low bandwidth and high latency. Today, the market offers countless competing video calling applications that benefit from the widespread adoption of broadband access and cellular networks. Each application employs different technical solutions and network protocols, although Real-Time Protocol (RTP) [2] is most widely adopted [3]. There are also efforts towards standardization, WebRTC being the notable example.[1]

In this context, it is essential to maximize the Quality of Experience (QoE) of users at the network level in order to avoid impairments, service misbehavior and consequently user churn. QoE depends on many factors, such as the quality of the participants' connection, network topology, and network management. Classification of RTC traffic is the first and most important step towards effective traffic management, allowing in-network devices to get an informed view of network flows and, if the classification is done in real time, to take appropriate actions to counter any degradation. The widespread adoption of encryption [4] has made it difficult for routers and middleboxes to separate traffic based purely on deep packet inspection (DPI) [5], while the convergence towards HTTPS has made port-based classification ineffective. However, most RTC applications adopt the RTP protocol [2] to encapsulate the multimedia content in its encrypted version, *Secure* RTP (SRTP). SRTP employs in-clear and straightforward packet headers, making its identification straightforward using existing DPI techniques. However, in SRTP, the media payload is encrypted, making it difficult to guess the type of content it carries. This calls for novel techniques based on machine learning (ML) to re-obtain visibility on application traffic and help decision-making at routers. For real-time communications, this could amount to distinguishing between top-priority flows and possibly less critical streams – e.g., audio as more important than video, the presenter's media as more valuable than the audience's media.

In this paper, we propose a novel ML-based application for classifying, in real-time, the RTP streams to the type of content they carry. Our approach is based on a few, but well-chosen features derived from the statistical properties of the traffic, which allow us to classify RTP streams using off-the-shelf supervised learning algorithms. Our approach identifies not only audio or video streams but also other properties of the media, such as the video quality or the use of Forward Error Correction (FEC) streams. Our solution works with minimal delay, deciding on the type of each stream within just 1 second of traffic. We design it as a software module that can

[1] https://webrtc.org/

be plugged into network devices (e.g., routers) or integrated into Software Defined Networks (SDN) to provide fine-grained traffic categorization and management.

Our study is based on two popular RTC applications for online multi-party meetings with audio, video, and screen sharing: Cisco Webex Teams[2] as a business-oriented platform and Jitsi Meet[3] as a lightweight in-browser application. Using data coming from more than 62 hours of real calls, we evaluate the impact of feature selection and different classification algorithms. After careful feature selection and using a lightweight decision tree classifier, we achieve an overall accuracy of 96% for Webex Teams and 95% for Jitsi Meet, with no large differences across classes. Our models require little traffic to train and do not introduce systematic errors. We note that models trained for one RTC application (e.g., Webex Teams) are hard to transfer to another application (e.g., Jitsi Meet) due to the different feature distributions. However, we show that we can partially overcome this limitation by using domain adaptation techniques.

Our proposal serves as a building block for a comprehensive traffic management system for RTC based on ML, enabling application-level visibility at the network control plane and providing highly detailed information about the running RTC sessions and the perceived QoE. This paper is the complete version of our previous work [6] and complements it in several directions. First, we motivate the work by showing scenarios of traffic management systems where our classification can be used. Second, we provide a thorough characterization of the dataset. Third, we extend our experimental evaluation to another RTC application (Jitsi Meet), as a representative of all applications used from the browser, thus doubling our experiments. We also discuss the transferability of the approach, whether we can use a model trained on one RTC application on another and how. Finally, we provide a thorough discussion on the importance of the features we used and how systematic the errors of the system are.

We make our dataset, code, and trained classifiers available online.[4] We believe they can help researchers reproduce our results or apply them to different contexts.

The remainder of the paper is organized as follows. In Section II, we describe the protocols and technologies we refer to throughout the paper, while in Section III we motivate our work illustrating the advantages of RTC-aware traffic management. In Section IV, we present and characterize our dataset, while in Section V we describe our methodology for feature engineering and classification. Section VI shows our experimental results, and, finally, Section VIII concludes the paper and discusses future work.

II. Background

To facilitate the reading of the paper, in this section we provide an overview of the most common protocols used in RTC applications and the difference between *native* applications (e.g., Webex Teams) and browser-based platforms (e.g., Jitsi Meet).

[2]https://www.webex.com/team-collaboration.html
[3]https://meet.jit.si
[4]https://smartdata.polito.it/rtc-classification/

Media streaming. The most common protocol for streaming media in real time is RTP [2]. Proposed in 1996, it defines a simple encapsulation mechanism in which different streams are multiplexed using a different Synchronization Source Identifier (SSRC). The RTP header contains a Timestamp field that reports the instant the content is generated and the Payload Type (PT) field indicating the video or audio codec used. RTP defines a set of predefined or static PTs and offers the possibility to define them dynamically during a session. RTP is carried over UDP or (very rarely) over TCP as a transport protocol. The control protocol RTCP is typically used alongside RTP to exchange various streaming statistics, such as packet loss rate. Secure RTP (SRTP) [7] is a variant of RTP that ensures confidentiality by encrypting the media payload while keeping all original headers in clear. In the remainder of the paper, we use the terms *RTP* and *SRTP* interchangeably.

Session Setup. In order to start a media session, it is necessary for the endpoints to be able to communicate with each other, especially in the case of peer-to-peer communication between participants. This is complicated by the presence of NATs, firewalls and middleboxes in general. To ensure connectivity, applications often use the STUN protocol [8] for NAT detection and TURN [9] to relay the traffic through a server with a public IP address. ICE [10] combines STUN and TURN into a single technique. RFC 7983 [11] defines a simple mechanism for multiplexing RTP, STUN, and other protocols on the same UDP flow.

WebRTC. The above protocols need to be carefully coordinated to have a working RTC application. To facilitate the development of new applications, WebRTC [12] is a set of high level and standardized APIs that can be used in browsers and mobile applications for video and audio communication. WebRTC was launched in 2011 and is currently supported by most browsers. It represents the standard way for RTC applications to run via web if we exclude application-specific plugins. WebRTC provides programming interfaces to establish media sessions, organizing the use of the RTP, RTCP, STUN, TURN and DTLS protocols.

RTC Applications Under Study. In our study, we focus on two RTC applications: (i) Cisco Webex Teams and (ii) Jitsi Meet. Webex Teams (or **Webex** for short) is a business-oriented service that offers paid plans for enterprises and institutions that require video call service. It is available as a standalone application for PC and mobile devices, but it can also be used through browsers that support the WebRTC standard. Jitsi Meet (or **Jitsi** for short) is a free of charge RTC application that provides a simple browser-based user interface for WebRTC-compliant browsers. It is fully open-source, and it is possible to run a private Jitsi server or rely on the public service available online. Important for our analysis, both applications use RTP for streaming multimedia content along with STUN and TURN for session establishment. They support audio and video communication and allow users to share their screens with the other participants. Moreover, they adopt the Selective Forwarding Unit (SFU) approach [13], where participants send their multimedia content to a central server. The server then forwards the data, deciding which

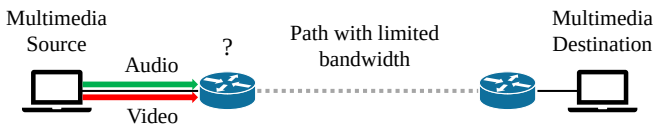


Fig. 1: Example of RTC-aware traffic management.

TABLE I: Experiment summary

Experiment	Media type	Packets lost	Packets received	Packet loss %
1	Audio	2136	6673	24,2
RTC-unaware	Video	4997	17031	22,6
2	Audio	0	8809	0
RTC-aware	Video	-	-	-

stream to send to each participant. Although the choice of different RTC applications (e.g., Zoom or Microsoft Teams) would be possible, we opted for Webex and Jitsi, which allow us to easily gather the classification ground truth, as we illustrate in Section IV. For other popular applications, we could not find such a convenient way to collect the needed information.

III. Motivation and Deployment Scenarios

In this section, we discuss the advantages of our proposal and possible deployment scenarios in real networks. We first illustrate how RTC-aware traffic management can practically improve user QoE, using a simple experimental setup. Our goal is to show that routing traffic not only based on the classical L3 packet headers, but also based on the media stream type, leads to sizable benefits under certain conditions. To this end, we setup a small testbed and assume our ML algorithm correctly classifies the media streams.

We outline our setup in Figure 1. Two hosts (the multimedia source and the destination) are each connected to a switch. The two switches are connected by a single path with limited capacity. The total available bandwidth for the path is 240 kbit/s. We set up a multimedia transmission where we send audio and video in two RTP streams. The audio is a high quality track with a bitrate of 140 kbit/s and the video has a constant bitrate of 200 kbit/s. The streams last 10 minutes and we send them simultaneously to emulate a video call. We build the testbed using the *Mininet*[5] tool to create the virtual network. We also use the Linux tool *tc netem*[6] to impose network constraints and *FFmpeg*[7] to stream the multimedia traces. To show the usefulness of our approach, we conduct two experiments: (i) *RTC-unaware*: the switch uses the classical approach to forward packets and thus treats both flows in the same way (ii) *RTC-aware*: our classifier is present and allows the switch to differentiate its behavior depending on the media type. Note that in both cases the required bitrate for both streams exceeds the capacity of the link.

[5]http://mininet.org/
[6]https://www.linux.org/docs/man8/tc-netem.html
[7]https://ffmpeg.org/

In both experiments, we quantify the impact on user QoE by using packet loss as a metric, since it has been shown to be closely related to QoE [14]. The results are summarized in Table I. In the *RTC-unaware* experiment, the switch drops packets from both audio and video streams, so we observe 24% of losses for audio and 23% for video. This renders the communication impossible, as such packet loss prevents audio and video streams from being decoded correctly. In the second experiment, *RTC-aware*, the switch detects that the bandwidth on the path is insufficient for both streams and therefore decides to forward the entire audio stream and discard the video stream instead of sacrificing both. In this scenario, the audio stream reaches the destination without any losses, so that the interlocutors enjoy at least audio communication.

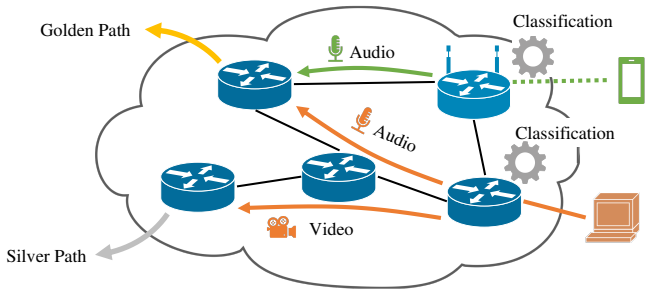
As this simple experiment exemplifies, our system can improve the QoE users perceive. Moreover, it enables RTC-aware traffic management so that scarce but expensive resources can be reserved for more valuable payloads (i.e., audio). Indeed, even in case we are unable to salvage the video, the network can preserve the audio. As proven in the literature [15], [16], when presented with a good audio and several different degraded versions of video, users perceive sufficient QoE.

On a general level, we foresee the use of our system in the context of RTC-aware network management and engineering, in which the network can make its decisions based on the type of multimedia content carried in streams. For example, the emerging SDN paradigm could benefit from RTC stream classification, allowing switches to steer traffic not only according to the classical protocol fields (i.e., addresses and ports) but also based on the media type of a stream. Similarly, in a Multiprotocol Label Switching (MPLS) network, the ingress node can set the label according to the classification outcomes. Also, approaches inspired to DiffServ or the IP *Type of Service* header are viable ways to differentiate traffic upon classification. Here, we do not explicitly target any of these possibilities but only show a few general cases where RTC traffic classification can be beneficial.

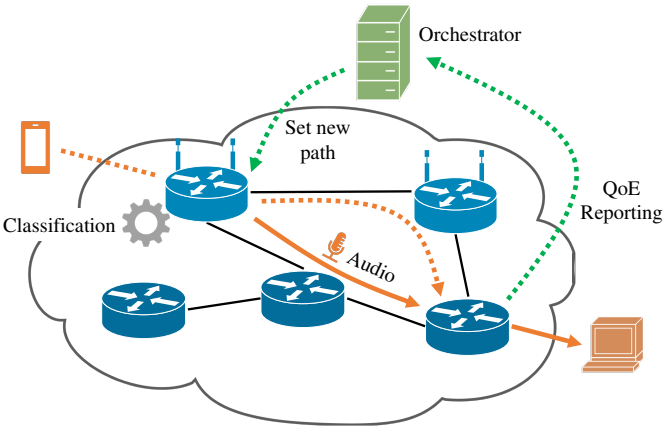
We sketch a first deployment scenario in Figure 2a. The edge network equipments run classification and select the path of each stream based on the media content they carry. In the example, audio packets are considered more critical and are routed to a *Golden* (reliable yet expensive) egress link, while the video is routed to a *Silver* (unreliable, yet cheap) path – e.g., a congested peering link.

We illustrate a second deployment possibility in Figure 2b. Here, our classification module is a building block of a more complex RTC-aware management system. Besides classifying streams to the content they carry (close to the source), network equipments report QoE-related metrics (close to the destination). The latter can be done in different ways, using well-known industrial standards [17], [18] or with other ML models [14]. A controller (or orchestrator) implements RTC-aware traffic management and can, like in our example, select new paths for RTC streams if it detects degradation in the measured QoE.

Note that the scenarios we envision are robust to possible flaws or delays in the underlying classification task. In Section VI, we report classification performance of 96.3% and



(a) Media-Aware Path Selection



(b) Path Selection based on media type and QoE feedback.

Fig. 2: Deployment scenarios benefiting from our classification system.

95.3% for Webex and Jitsi, respectively, with a delay of 1 second for collecting statistics and a few milliseconds for computing features and running the classification. Now we evaluate the impact on our proposed deployment scenarios, taking into account that both proposals (Figure 2a and Figure 2b) work by promoting streams to a better path when QoE is poor or the link is congested. There are generally two types of misclassification. In the first case, the error causes the system to respond unnecessarily – for example, we classify a video stream as an audio stream and promote it to a more reliable path. In this case, the system wastes resources unnecessarily. In the second case, the error does not trigger a system response when it should have – e.g., we classify a stream that is actually an audio stream as video and do not promote it. In this case, the system would maintain the *status quo*, i.e., a "bad" QoE. In this sense, an accuracy of 95-96% means that the system improves the QoE in 95-96% of the cases, while in 4-5% of the cases we maintain the status quo or we waste some resources. Although undesirable, these situations do not entail severe impairment in QoE or in the whole system, provided they are sufficiently sporadic.

As for delay, we believe a delay in system reaction in the order of 1 s is tolerable for video calls, since their lifetime is in the order of minutes or hours. Collecting information about a stream for 1 second allows us to compute representative statistics about the stream, thus increasing the accuracy of the classifier. In Section VI, we also show that it is possible to use our classifier with slightly worse accuracy at a reduced delay of 200ms.

TABLE II: Dataset summary

Class	No. of seconds			
	Webex		Jitsi	
	Train	Test	Train	Test
Audio	224 295	80 781	123 745	30 180
Video LQ	200 380	76 825	84 134	20 192
Video MQ	55 112	18 156	34 708	7 817
Video HQ	59 073	19 526	33 049	7 920
Screen Sharing	41 170	8 800	29 216	6 870
FEC Audio	146 567	41 247	-	-
FEC Video	45 591	2 164	-	-

IV. Dataset

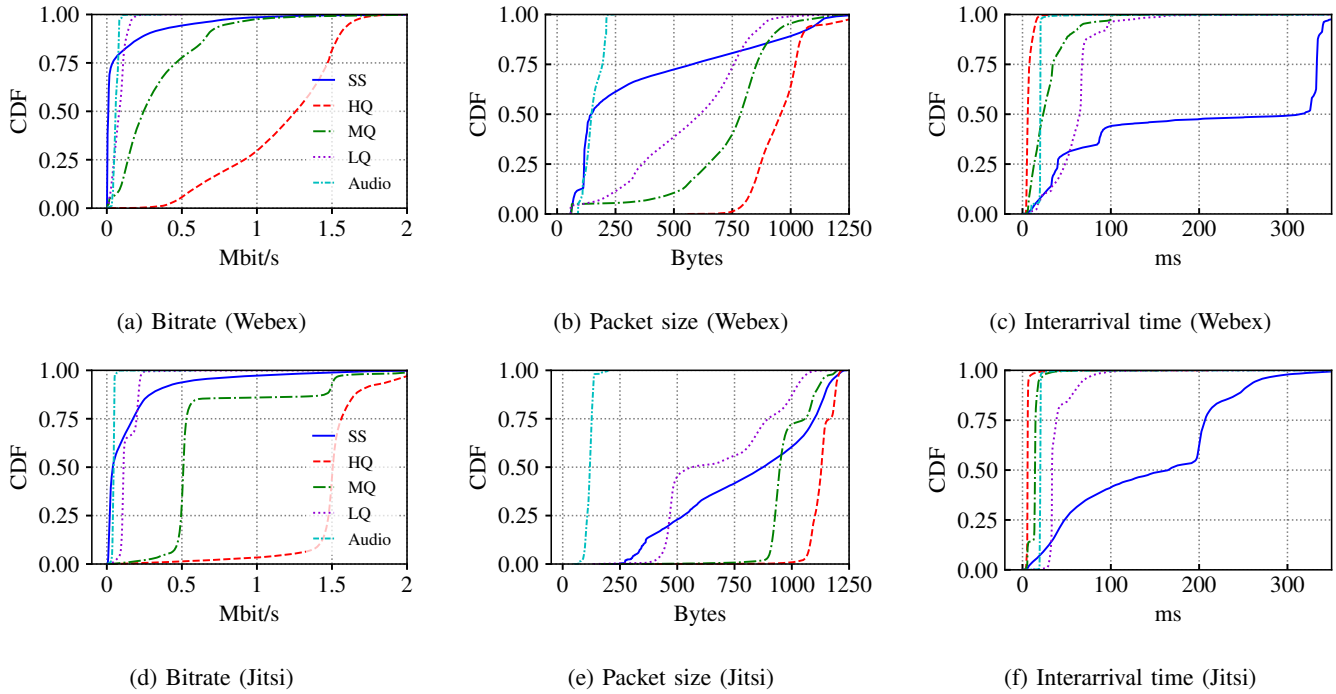
A. Data collection

In this section, we describe the dataset we use throughout the paper. We target the two RTC applications described in the previous section, namely Webex and Jitsi. With both applications, we capture real calls made under different conditions, with a different number of participants (from two to ten), multimedia content (audio, video, screen sharing), and user equipment (PC, tablet, or phone). The calls run in a real environment where participants are connected via different networks from 3 countries and use different devices, from Windows PCs to iPhones and Android phones. During each call, at least one participant captures all the exchanged traffic and stores it in `pcap` format. The calls took place over a period of 6 months.

In our classification problem, we target RTP streams, which we identify with the tuple: source IP address, source port, destination IP address, destination port and RTP SSRC. In other words, we target a single stream that carries a specific multimedia content. We divide the streams into 5 classes: Audio, Low Quality (LQ) Video (180p), Medium Quality (MQ) Video (240-640p), High Quality (HQ) Video (720p), and Screen Sharing. For Webex, we consider two additional classes: FEC audio and FEC video. Indeed, Webex uses FEC to mitigate packet losses, sending streams with redundant information to be used at the receiver if some packets are lost or contain errors. We observe FEC streams for audio and video, and we are interested in identifying them as separate classes. Hence, seven classes are considered when analysing Webex data.

We employ the debugging logs to gather the ground truth, which maps each RTP stream to the content type. For Webex, logs are automatically generated during each call, while for Jitsi we use the Chrome browser WebRTC logs.[8] The logs for both applications contain per-second statistics for each stream, including the type of media (audio, video or screen sharing),

[8]This log can be obtained by creating and downloading a dump at *chrome://webrtc-internals*



(a) Bitrate (Webex) (b) Packet size (Webex) (c) Interarrival time (Webex)

(d) Bitrate (Jitsi) (e) Packet size (Jitsi) (f) Interarrival time (Jitsi)

Fig. 3: Distribution of traffic characteristics for Webex (top) and Jitsi (bottom), separately for media stream type.

the video resolution and the number of frames per second. During each call, the participant who captures the traffic also collects the logs, which we store alongside the `pcap` trace. Note that we cannot use the RTP Payload Type field for this, as it is dynamically assigned.

We collect traffic for approximately 62 hours of video calls, exchanged during 27 meetings with Webex and 50 meetings with Jitsi. They sum up to 90 GB of `pcap` files, which include the call traffic as well as a small amount of background traffic that we neglect. The dataset contains 3 977 RTP streams for Webex and 521 for Jitsi. Each call contains a different mix of the above classes, and includes traffic generated by all participants as captured from the point of view of a single individual. Out of the 77 calls, 35 have only two participants, 11 have three participants and 31 include more than three. In Table II we give an overview of the dataset, separating the training and test set. In Section V we describe our training/testing methodology in detail. For each RTC application and class, we report the amount of data we collected, in seconds. The most represented classes for both applications are Audio and LQ video. While this is somewhat expected for audio, the prevalence of LQ video is due to the video thumbnails used in the applications to show inactive participants during calls with more than three participants. Note that for Webex, FEC audio is also widely represented. The least represented class is Screen Sharing, but the overall dataset imbalance is still limited, with the ratio between the support of the most and the least represented class being less than 6.

B. Characterization and challenges

We provide a high-level overview of the dataset in Figure 3, where we plot the Cumulative Distribution Functions (CDFs) for different stream features, separately by application. We use different lines to contrast the four video-based classes, plus audio. The leftmost figures show the bitrate distribution for Webex (Figure 3a) and Jitsi (Figure 3d). For each stream, we compute the average bitrate using 1-second bins. We first note that better video qualities tend to have higher bitrates (e.g., red and green lines). Audio (cyan line) has the lowest bitrate, as expected. However, the two applications present different shapes for the video curves. Webex displays smooth distributions, indicating that it adjusts the target bitrate of the video codec. In contrast, Jitsi exhibits a cascading behaviour, indicating thresholding and somewhat quantized bitrates. Note that the same video quality appears with multiple evident bitrate peaks. For example, MQ video (green dashed line) presents two peaks roughly at 0.5 and 1.5 Mbit/s, both corresponding to 640×360 video. The Screen Sharing class (solid blue line) exhibits the greatest variability. Again, this is expected, as it carries diverse contents, from slide sharing to scrolling through the screen, to effectively playing a video. This leads to a generally low bitrate with short periods of high activity. We note that setting a simple threshold on the bitrate would not yield accurate class predictions. This is especially true for Webex, where the distributions overlap significantly. In particular, for screen sharing, the bitrate ranges from a few kbit/s to more than 1 Mbit/s. Interestingly, the Screen Sharing bitrate is often as low as an audio stream, for both applications.

Similar considerations hold for the packet size (Figures 3b and 3e). Better video qualities tend to use larger packets as they sustain a higher bitrate. Again, we observe a high overlap of Screen sharing with all other classes. For Webex (Figure 3b), Screen Sharing packets can be as little as those of audio streams. Conversely, for Jitsi (Figure 3e), only audio

uses small (100-150B) packets, potentially easing its identification.

Finally, the rightmost figures show the distribution of packet inter-arrival time for Webex (Figure 3c) and Jitsi (Figure 3f). We compute it as the time interval between two consecutive packets in the same RTP stream. The video distributions partially overlap, with Screen Sharing presenting inter-arrival time as large as 400 ms when nothing on the screen is changing. Figure 3 shows that a careful mixture of these features is required for accurate prediction. In the remainder of the paper, we show that it is possible to identify the type of media stream with high accuracy using features derived from these traffic characteristics and a machine learning classifier.

V. Methodology

In this section, we describe the proposed approach, from RTP traffic identification to feature extraction and classification. We envision an offline training of a classification model and its application to live traffic in real-time. We sketch a high-level overview of our approach in Figure 4. We also describe in detail the methodology to build and select the features from RTP traffic. We follow the same approach for both Webex and Jitsi and create a separate classifier for each. Throughout this section, we use Webex as a running example to facilitate the understanding of the methodology.

Problem statement. Our goal is to classify the RTP streams that we observe on the network to one of the classes listed in the previous section and Table II. We want to solve this task in real time, i.e., make a decision based solely on the traffic observed in a short time interval, by applying a model trained on historical data. Thus, our classification target is an RTP stream as observed during a certain time bin (from 200 ms to 5 s).

RTP stream identification. We identify the RTP traffic with straightforward deep packet inspection, by matching the protocol headers. Indeed, the RTP header includes fixed-sized fields that facilitate its identification, and its sequence number serves as a simple sanity check for identification, since it must increase by 1 for subsequent packets. Popular passive meters identify RTP flows using DPI – e.g., Tstat [19] or nProbe [20]. Note that we do not handle the case of RTP tunneled through an encrypted channel (e.g., over a VPN or IPSec tunnel), since we cannot distinguish the different streams. We separate multiple media streams via their SSRC. We are not interested in the control traffic for, e.g., session establishment or login, and thus neglect it. We also assume that we know the application in use (Webex or Jitsi), since different techniques may be used for this purpose. In some cases, RTC applications provide public lists of the relay server IP addresses or use well-known ports [3]. Webex, for example, uses UDP port 5000 for RTP streams. In case such an approach is not feasible, it is possible to leverage ML-based solutions. In our previous work [21], we showed how to guess the RTC application in use with high accuracy using the domain names that the client resolves over the DNS prior to the call and an ML classifier.

The ML pipeline. A single RTP stream results in many samples (one per time bin) that we shall classify. For our classification problem, we follow the classical approach of supervised learning. First, we extract meaningful features from the data, guided by domain knowledge on network traffic and RTP protocol. Then, we perform a two-step feature selection process by first discarding highly correlated features and then performing a recursive feature elimination. Finally, we train a machine learning classifier and evaluate its performance on an independent test set. Feature selection and algorithm training are performed offline, while the system is designed to compute features and classify new samples in real time. The time it takes is equivalent to the chosen time bin plus the feature computation and algorithm run, whose execution time is negligible. Our code is written in Python and uses the scikit-learn library [22] for machine learning. Our methodology is readily amenable to parallelization, as all processing is done on a per-flow basis – i.e., feature extraction and classification only need to obtain data from a single stream. Therefore, a multi-core parallel approach is fully feasible, and we do not expect any bottlenecks in high-speed deployments, provided packet capture is adequate. In case of deployment with off-the-shelf hardware, high-speed packet capture libraries (e.g., DPDK[9]) together with Network Cards natively supporting load balancing (e.g., Receive-Side Scaling on Intel cards) would perfectly serve at this goal.

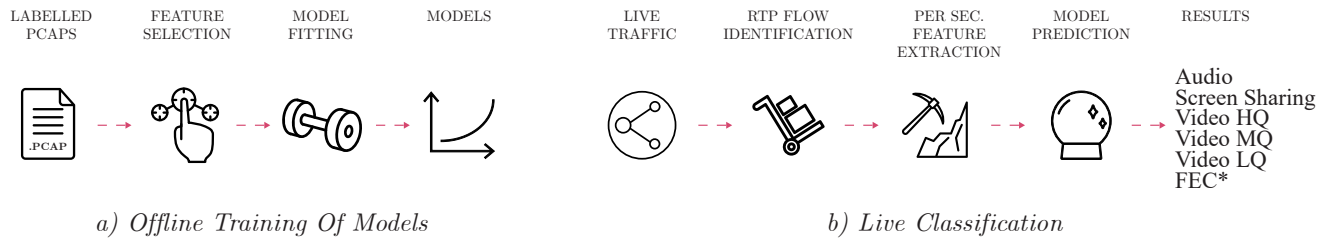
Train/test methodology. We split the call dataset into a training and a test set to prevent overfitting and obtain robust results. We perform feature selection and algorithm hyper-parameter tuning on the training set, and we evaluate classification performance on the test set (which we never use at training). Note that the streams of a single call are used either at training or testing time to keep the two sets completely independent. For Webex, out of 27 calls, we use data from 22 calls for training and data from the remaining 5 calls for testing. For Jitsi we use data from 41 for training and 9 for testing. With this split, we obtain roughly 80% of samples (1-second bins) for training and 20% for testing (see Table II). We also verify that each class is well-represented in both sets. As a global performance indicator, we use the macro-average (a simple mean) of the F1-scores of each class.[10] For some analyses, we also consider accuracy as a concise index of overall performance, since classes are not strongly imbalanced.[11]

Feature extraction. We extract features from the packets separately by RTP stream and time bin. The features are based on the fields of the RTP protocol and take into consideration its operation. We outline our approach in Figure 5. We consider five groups of features, reported in the middle column of the figure, in bold. These include packet characteristics (size, time, volume) and the RTP timestamp field, which indicates the time at which the content was generated at the source. RTP has a few other fields that essentially indicate header

[9] https://www.dpdk.org/

[10] The *F1-Score* is the harmonic mean between $Precision$ and $Recall$ of a class.

[11] The *accuracy* is the share of correct predictions over the total.



a) Offline Training Of Models *b) Live Classification*

Fig. 4: Overview of the training and classification pipeline.

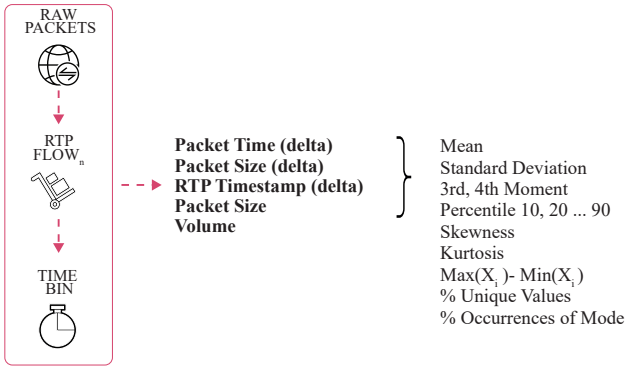


Fig. 5: Features derived from packets.

extensions, which we do not include because they are very application and client-specific. Since two of the selected fields (packet time and RTP timestamp) represent time instants, we only consider their relative variation across packets, since the absolute values are useless in our context. For packet size, we use both absolute and relative values. We extract these five values for all packets and compute various statistical indices to create the final features, such as range, mean, standard deviation, percentiles, third and fourth moments, etc. Since we find that the same values recur frequently in the packets, we also add features that measure the number of unique values, the percentage of occurrence of the most frequent value (mode), and the ratio between the minimum value and the range. We report the complete list of statistical indicators on the last column of Figure 5. Finally, we consider the traffic volume in terms of the number of packets and bitrate observed in the time bin. We also use the number of packets with the RTP marker flag set as a separate feature.

Since our goal is to design a real-time classification system, we create features that can be computed on the fly by considering only the packets observed in a time bin. Intuitively, the smaller the time bin is, the faster the stream is classified. However, features are more representative with larger time bins since they are computed over a more extensive set of packets. In Section VI, we explore this trade-off and evaluate how the temporal granularity affects the classification of an entire stream. Finally, note that we also avoid features that require linking multiple streams to keep our design simple and easily parallelizable.

Feature selection. In total, we extract 96 features derived from the four empirical distributions mentioned above, plus volume.

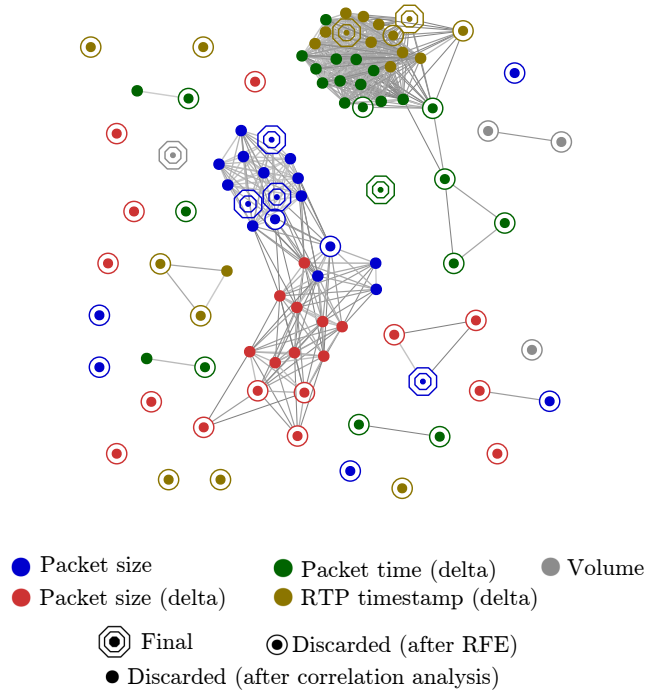


Fig. 6: Graph representing the correlation between features. The color indicates the feature set, the shape whether the feature is kept after feature selection and the distance represents the correlation.

We publish the full list of features on our research center website.[12] To remove those that are redundant and shrink the overall number of features, we perform a two-step selection process.

1) *Correlation analysis*: We perform an initial feature selection by measuring the correlation between each pair of features. We evaluate all possible pairs in a random order, and whenever we find a Pearson correlation coefficient greater than 0.9 (in absolute terms), we keep only one of the two features at random. With this step, we roughly eliminate half of the features.
2) *Recursive Feature Elimination using the ExtraTree algorithm*: We use the Recursive Feature Elimination (RFE) approach [23] to refine our list of features, maintaining only those that are most useful for our classification problem. Using RFE, we train an ExtraTree classifier

[12]https://smartdata.polito.it/rtc-classification/

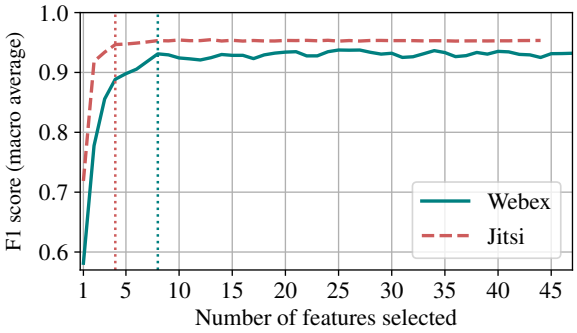


Fig. 7: Mean F1 score when varying the number of features. The vertical lines indicate the final number of features.

on the training set and rank the features by their *feature importance* as provided by the algorithm.[13] We then eliminate the one with the least importance. We recursively repeat this procedure until we reach the minimum number of features and the best performance, which we evaluate using 5-fold cross-validation. Note that tree-based feature ranking is known to be biased in the case of groups of correlated features [24]. Thus, our first step (correlation analysis) is essential for RFE to work correctly.

We graphically illustrate the entire feature selection process for Webex with Figure 6, which shows the initial 96 features in the form of a graph. Each node represents a feature, and the length of edges is (roughly) inversely proportional to the correlation among pairs in absolute value – i.e., highly correlated features remain close to each other. For illustration purposes, we only show the edges where the correlation is higher than 0.5 (in absolute value). Different colours represent the feature sets, while the shape of each node indicates whether a feature is maintained or discarded at one of the selection steps: a circle means that the feature was discarded after correlation analysis, a double circle means that the feature was discarded with RFE, and an octagon means that it passed both steps and is included in the final list.

We first notice that the correlation analysis step maintains all features which are poorly correlated with other ones: all nodes without edges are either double circles or octagons. On the contrary, among groups of highly correlated features, only a few samples are retained. For example, the dense community in the top right of the figure includes the percentiles of packet time inter-arrival time and RTP timestamp, which are intuitively highly correlated. We retain only two of them.

Continuing with the running example of Webex, the first step of the feature selection shrinks our set from 96 to 47 features. We then perform RFE to obtain only those that are useful for our classification problem. We train an ExtraTree classifier on the remaining 47 features, running a 5-fold cross-validation to evaluate how accurate the obtained model is. We then eliminate the feature ranked as least important and repeat this process until we find that the classification performance starts to decrease. In Figure 7, we show how the average F1 score varies when removing an increasing number of features. The figure shows our results for both Webex (solid blue line) and Jitsi (red dashed line).

Considering Webex, when we use all 47 features, we get an F1-score of 0.91. The performance is almost stable (with minimal variations) until we use 8 features only – i.e., we eliminate 39. Then, the accuracy starts decreasing consistently. After analysing the curve, we decide to set the final number of features to 8. Interestingly, we notice that every feature group (except the packet size) appears in the set of the final features (there is an octagon of every colour except red in Figure 6). Among the final features, we find the packet size (mode, 25th, 70th and 75th percentile), the 30th percentile and mode of the RTP timestamp delta, the mode of the inter-arrival time and the number of packets with the RTP marker flag set. Intuitively, for each characteristic of the packets, we keep a few statistical properties of its distribution.

The process is similar for Jitsi (red dashed line in Figure 7). Note that the curve ends at 43 features, since for Jitsi the first step of feature selection eliminates a slightly larger number of features. The knee in the line shows that we already achieve good performance with as little as 4 features. Among them, we find three representatives of the packet size feature group and the mode of the RTP timestamp delta. This indicates that the packet length is a vital factor for this classification problem.

Multi-class classification. Using the features that we obtain after the feature selection, we try different classification algorithms to find the one that yields a proper trade-off between performance and simplicity. The algorithms we consider are: tree-based classifiers [Decision Tree (DT) and Random Forest (RF)], k-Nearest Neighbors (k-NN), which classifies points based on proximity to other data points, and Gaussian Naïve Bayes (GNB) as a generative probability model. We perform hyper-parameter tuning with 5-fold cross-validation for each of these models, using the training set uniquely. We then evaluate their performance on the separate test set, using the macro-averaged F1-score as a performance indicator. In Section VI, we show that the algorithm choice has a moderate impact on classification performance.

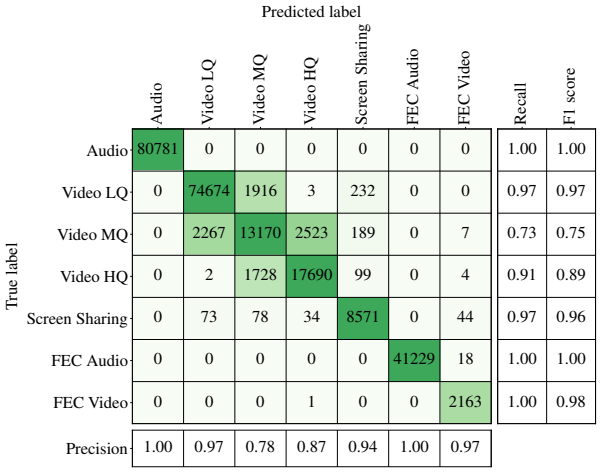
VI. Experimental Results

In this section, we present our experimental results for the entire classification problem. First, we discuss the overall classification performance and quantify the impact of the time bin duration, classification algorithm and training set size. We then discuss the importance of the features and analyze how classification errors arise. Finally, we investigate the possibility of transferring a model trained for one RTC application to another. All results are obtained by training classification models on the training set and evaluating their performance on the independent test set.

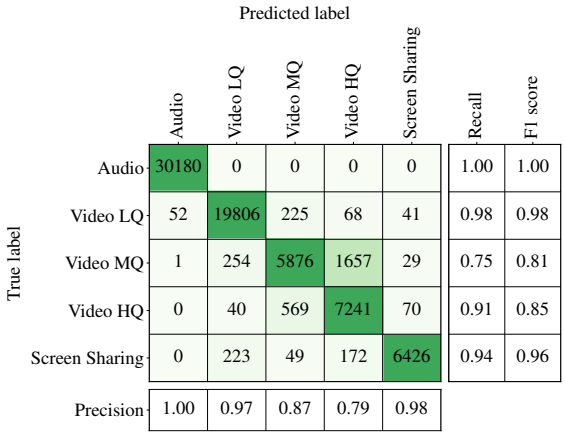
A. Classification performance

We first report and discuss the performance we obtain for both RTC applications when using the best models. Indeed, we try different classification algorithms and finally opt to use a

[13]The ExtraTree classifier natively exposes the feature importance after training.

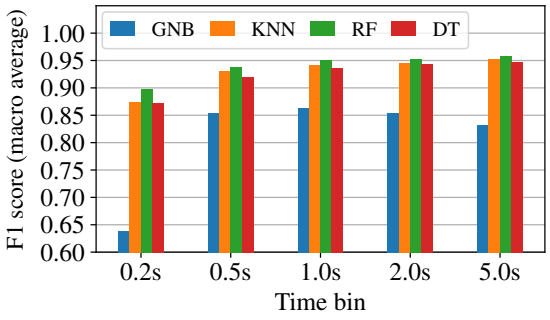


(a) Webex

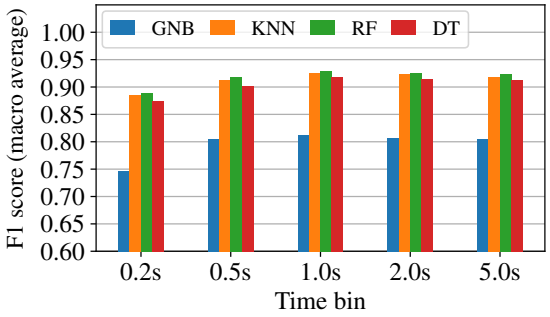


(b) Jitsi

Fig. 8: Confusion matrices when using a Decision Tree classifier and 1s time bins.



(a) Webex



(b) Jitsi

Fig. 9: Performance of the four algorithms for different time bins.

Decision Tree classifier, which provides good performance and a simple model. Running hyper-parameter tuning, we obtain the best results when using the Gini index as a purity measure. In Figure 8, we show the confusion matrices for both Webex and Jitsi using a 1s time bin. By definition, a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j. Thus, the main diagonal represents the number of correctly classified samples. We also show the per-class recall and F1-score in the last two columns and precision in the bottom row. We note that, for both applications, all classes except Video MQ and HQ exhibit an F1-Score above 0.96, and thus high precision and recall. Audio is the best performing class for both RTC applications, together with FEC audio for Webex. Here only a handful of samples are misclassified, suggesting that audio streams are generally easy to isolate. Indeed, for Jitsi especially, audio streams tend to use smaller packets than video (see Figure 3), making their identification simpler. The worst performing class is video MQ, with F1 scores of 0.73 and 0.75 for Webex and Jitsi, respectively. The confusion matrices reveal that the three different video qualities are, in some cases, confused with each other. Although this is a flaw of our classification model, we tolerate this behaviour given the similar nature of the three classes. Also, keep in mind that applications (especially Webex) use video codecs with variable bitrates that result in different network traffic (see Section IV). Overall, for Webex, 96.3% of the samples are classified correctly (i.e., accuracy), and the average F1-score is 0.94. For Jitsi, we obtain an accuracy of 95.3% and an average F1-score of 0.92.

Considering computational time, our system needs to perform 3 consecutive steps before providing the final classification label: (i) Wait for the time bin to gather traffic information, (ii) Calculate the features and (iii) Apply the classification model. Step (i) obviously takes most of the time. Step (ii) depends on the class, with Video HQ being the most expensive as it sends the highest number of packets, thus increasing the number of samples in the calculation. On average, this step takes few milliseconds with our Python code on commodity servers. Finally, step (iii) is even faster, requiring the use of a light-weight decision tree model, that takes tens of microseconds. For high-speed deployments, we envision the use of a parallel multi-core architecture to scale the processing. Such an approach is completely feasible since the classification relies on features extracted on a per-UDP flow basis.

B. *Parameter sensitivity*

We now discuss the impact of the time bin duration on the classification performance. Indeed, we are interested in classifying a stream as fast as possible without sacrificing accuracy.

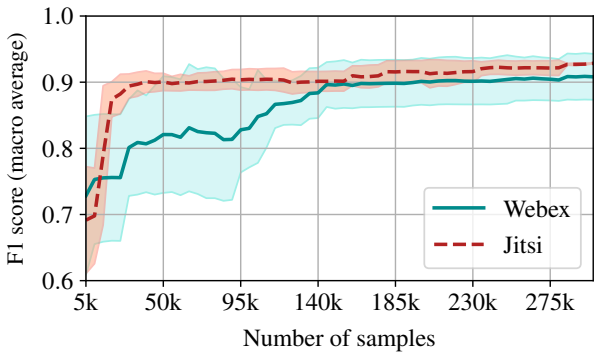


Fig. 10: Learning curve: Relationship between the number of training samples and the score.

Figure 9 shows how performance varies with different time bin durations, from 200ms to 5s. We provide results for 4 classification algorithms, and the y-axis reports the average F1-score we obtain. We find that we generally get better results with larger time bins. This is no surprise since the features are computed over more extensive sets of packets. For example, in 200ms of a typical audio stream, only 10 packets are generated. The performance flattens for values larger than 1s for both applications, implying that such a time frame is large enough to capture representative features about a stream. We believe that a delay of 1s is not critical, since RTC calls typically last minutes.

Looking at Figure 9, we can also compare the performance of different classification algorithms. We observe no large differences, except for Gaussian Naïve Bayes, which exhibits somewhat worse performance, probably due to the simplicity of the model. Note that the lowest F1-score is 0.62 for Webex and 0.73 for Jitsi. This confirms that our careful feature engineering and selection make the results robust to the choice of algorithm. We finally opt to use a Decision Tree for its simplicity, interpretability and speed. Random Forest produces similar results, but is more computationally intensive as it uses trees in parallel, 100 in our case. k-NN also performs well, but requires the model to store the entire training set in the main memory, resulting in significant memory consumption. Using a decision tree instead, the model is only a few kB in size. Comparing the two applications confirms that they exhibit very similar performance, with Jitsi having a lower F1-score by about 0.02 in most cases.

C. Training set size

We now investigate how much training data is necessary to achieve good classification performance. To this end, we train many classification models, gradually increasing the size of the training set. We vary the number of training set samples selecting them from the least possible number of calls. In other words, we entirely consume the samples from one call before drawing them from a second. In this way, we indirectly observe how many calls are required. Note that randomly selecting training data from all calls would likely sample the diversity of the entire dataset, which is unfair for our analysis. In this experiment, we use Decision Tree classifiers with 1s time bins.

Figure 10 shows the classification performance versus the training set size. Again, we measure the performance using macro-averaged F1- score on the test set. We repeat each experiment 5 times, shuffling the order of the calls but still drawing samples from one call altogether. The solid blue and red dashed lines indicate the mean score of the experiments for Webex and Jitsi, respectively. The areas represent the standard deviation across the runs. Starting from Jitsi, we notice that the performance improves very quickly with the training set size– with only 20k samples, the F1- score is already above 0.86. Such an amount of time corresponds to 5 hours of audio and video call. After that, it increases very gradually, reaching a local maximum of 0.92 F1 score at 200k samples (55 hours of calls). The standard deviation is generally small and stable. This result suggests that the features we extract and the nature of the problem do not require a large dataset to obtain a reliable model. Conversely, Webex requires a larger training set for accurate classification, exhibiting a slow growth and a larger standard deviation, stabilizing at 145k samples (40 hours of calls). This is likely due to the higher number of classes (with the additional audio and video FEC classes) and a variegated behaviour of the application within a call. Indeed, we observe that there is an abundance of audio and video LQ in various calls and a deficiency of the other classes. Consequently, additional calls are necessary to bridge the gap. To test this conjecture, we perform an additional experiment where we balance the number of samples per class and find that the performance converges faster.

D. Feature analysis

We now discuss the outcomes of the feature selection phase. Our goal is to investigate whether we can recommend a fixed set of features for any RTC application or they are specific for each one. As described in Section V, we carry out a two-fold feature selection: we first remove highly correlated features, and then we perform recursive feature elimination using an ExtraTree classifier. In Figure 11 we compare the results of the second step for Webex and Jitsi. Each symbol represents a feature that we retain after the correlation analysis – 43 for Jitsi (upper row) and 47 for Webex (lower row). Circles represent the features that are finally selected, and their size is proportional to the relative importance given by the ExtraTree classifier. The squares represent the remaining features, that were discarded using RFE. We arrange them in the order in which they were discarded. The colours indicate the feature group, similar to Figure 6. The edges connect the same feature on the two RTC applications so we can compare Jitsi and Webex.

As anticipated in Figure 7, with Jitsi, 4 features are enough to achieve good performance, while Webex needs 8. Looking at Figure 11, we observe a large presence of features related to the packet size (blue) – 3 out of 4 for Jitsi and 4 out of 8 for Webex. This is expected, as the packet size is instrumental for distinguishing audio and video streams (see Figure 3). We note that 3 of the Jitsi features also appear in Webex, albeit with

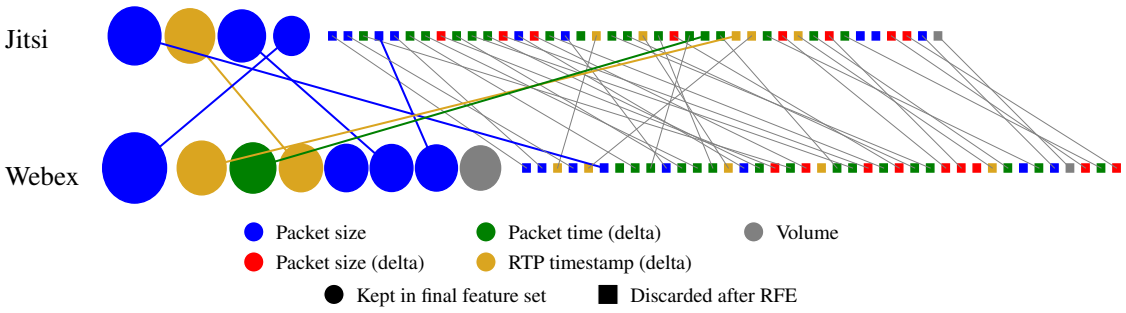


Fig. 11: Feature importance comparison between Webex and Jitsi.

different importance. Overall, the features are ranked similarly for the two applications, and the Spearman's rank correlation coefficient between the two ranks (including all features shown in Figure 11) is 0.70. Interestingly, two features chosen for Webex have been discarded in the first feature selection phase for Jitsi – two circles on the bottom row are not connected to any of the above shapes. A notable one is the number of packets with the RTP packets with the marker flag set (the gray circle). We note that this feature correlates strongly with frame rate in video streams, and speculate it helps identify the screen sharing class, typically with a low frame rate.

E. Error analysis

We now analyze misclassification cases to understand (i) how they are spread among streams and (ii) whether they can affect the prompt classification of streams.

Overall, we obtain an accuracy of 96.3% for Webex and 95.3% for Jitsi, as detailed in Section VI-A. Here, we want to measure whether these errors are concentrated on a few RTP streams or are scattered between all. To this end, in Figure 12, we plot the complementary cumulative distribution function (CCDF) of the percentage of errors per RTP stream. In other words, for each stream in the test set, we compute the percentage of misclassified samples and then show the distribution over all streams. The test set includes 508 streams for Webex and 101 for Jitsi. We observe that most of them present a rather low error rate. For Webex (solid blue curve), we notice that the probability of misclassifying more than 10% of the samples of a stream is $\approx$ 10%. Moreover, the probability of misclassifying more than 50% is less than 2%. This result suggests that, in general, mistakes span through many different streams rather than all originating from a few, and our classifier typically does not commit systematic errors. Similar considerations hold for Jitsi. There are only a handful of streams for which most samples are assigned to the wrong class – see the right-most side of the plot. These are usually short-lived streams (shorter than $10s$), except two long Webex video MQ streams where 68% of samples are misclassified and one long Jitsi video MQ stream with 73%. As reported in Section VI-A, video MQ is the hardest class to discern. In conclusion, these results show that the misclassification of an entire flow is very unlikely to happen.

We now investigate the possibility of classifying an entire stream just by looking at the first few samples. It might be beneficial in some real deployments when the network must react quickly to new streams to – e.g., prioritize particular traffic classes (see Section III for possible deployment scenarios). To this end, we suppose to classify a new *stream* based on the first N samples, using a majority vote scheme on the labels we obtain for those samples. In other words, given the first N samples of a stream, we assign it entirely to the class most samples have been assigned to. In Figure 13, we show the macro-averaged F1-score we obtain, varying N between 1 and 30 seconds. In this case, the classification goal is a *stream* rather than a *sample*, and, as such, we compute performance metrics over the streams in the test set. When classifying the stream based solely on the first second, we obtain 0.92 macro-averaged F1-score for Webex (solid blue line) and 0.82 for Jitsi (red dashed line), as sporadic errors have the maximum impact. Increasing the number of samples N, we obtain better results, reaching macro-averaged F1-Score of 0.99 and 0.93 for Webex and Jitsi, respectively. Indeed, our classifier hardly perpetrates systematic errors (see the previous paragraph), making the majority voting scheme very robust to misclassification. We conclude that our approach is fully appropriate in contexts where the network is required to quickly make decisions on an entire flow, e.g., installing appropriate SDN rules on the network switches.

F. Model Transfer To Other Applications

In our previous results, we train a classifier with labelled data belonging to the same RTC application that we aim at classifying. This might not always be possible, as labelled data are hard and expensive to obtain. Moreover, new RTC applications may spread rapidly without controlled experiments being possible. In this section, we explore to what extent a classifier trained for RTC application A can be used to classify streams of the application B.

For our goal, we investigate the use of *transfer learning* techniques [25], whose goal is to transfer knowledge from one *domain* (i.e., one RTC application) to another. These techniques are useful when we cannot collect labelled data in the second domain. In this case, we can try to use the knowledge from domain A to solve the same problem in domain B. In general, the rationale behind transfer learning techniques is to modify and adapt an ML classifier trained in domain A to classify samples in domain B.

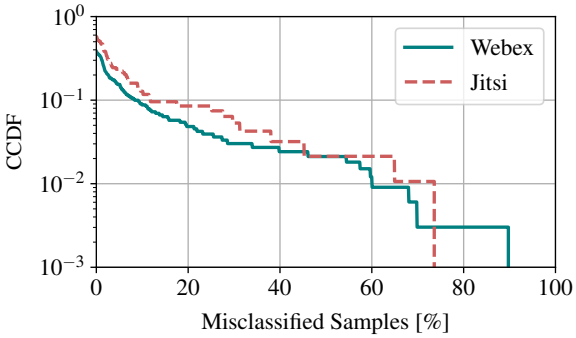


Fig. 12: CCDF of percentage of errors per stream.

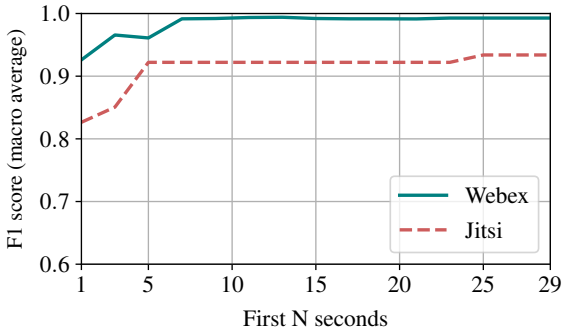


Fig. 13: Classification performance using first N samples per stream.

Here, we employ the domain adaptation technique called CORrelation ALignement (CORAL) [26]. As the name suggests, given the feature distributions from two domains (A and B), CORAL tries to align the covariance matrix (matrix of second-order moments) of distribution B to the one of distribution A. Due to the nature of our problem, we hypothesize this approach suitable since we target two similar RTC applications that use the same network protocols. Necessary for our goal, CORAL is an unsupervised technique, as it assumes data for domain B are available, but without class labels.

We here investigate the performance we obtain when using a classifier trained on application A (e.g., Webex) for classifying data of application B (e.g., Jitsi). We perform experiments (i) using the classifier directly on application B and (ii) using CORAL to align domains A and B. Case (i) corresponds to using a classifier directly outside of the training context. In case (ii), we assume that non-labelled data for application B are available, allowing the use of CORAL to align the two domains. We show the results in Figure 14, again measuring performance in terms of macro-averaged F1-Score. The x-axis reports the domain on which the classifier is trained, while the colour of the bars indicates the domain on which we use it. We provide a reference using the green bars, indicating the performance we obtain when we use the classifier in its domain –i.e., the approach we used in the previous sections. For this experiment, we remove the FEC streams from the Webex traffic, since we need the same number of classes

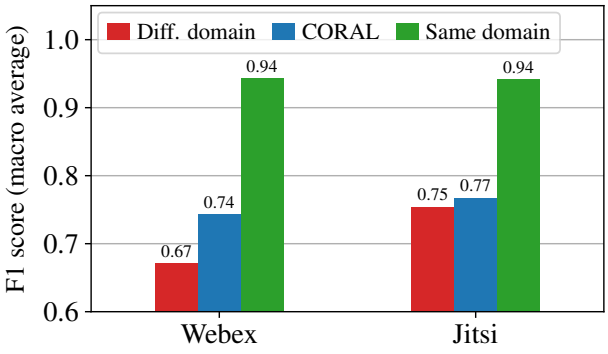


Fig. 14: Classification performance varying the target domain.

for the two applications for a fair comparison. The red bars represent case (i), while the blue bars case (ii).

We first notice how using a classifier directly on a different RTC application entails a certain performance drop (red bars). Indeed, using a classifier trained on Webex to classify Jitsi streams leads to a 0.67macro-averaged F1 score. In the opposite direction (training on Jitsi and testing on Webex), the performance is slightly better (0.75). The use of CORAL improves the performance in both directions, yielding similar results in both directions (blue bars). We get an F1-Score of 0.74 when training on Webex and using Jitsi and 0.77 vice-versa. Interestingly, the benefit of CORAL is higher in the former case (+0.07), while minimal in the second (+0.02). Nevertheless, it is still far from the performance obtained by training a model on the same domain, which then soars to an F1-score of 0.94 for both applications (green bars). This might originate from the different shapes of traffic distributions between the two RTC applications, as discussed in Section IV. In summary, our results suggest that it is possible to use a classifier for a different application if lower performance can be tolerated. If non-labelled data for the target RTC application are available, CORAL is instrumental in increasing the performance.

VII. Related Work

Network traffic classification has been extensively studied since the birth of the Internet [5]. Due to the widespread adoption of encryption and the use of proprietary protocols, traditional approaches based on mere DPI and port numbers fall short, and the current research tends to use statistical traffic features and machine learning techniques [27]. Recent efforts aim to identify the web services [28] or mobile applications [29] behind network traffic, predicting the QoE of web [30], video [31] or smartphone [32] users.

In Section VI-F, we investigate the use of transfer learning techniques for our classification problem. A few works already proposed their use for problems related to networking, albeit in different contexts. Authors of [33] use transfer learning in wireless networks for a caching procedure. Instead, the approach proposed in [34] used it in combination with Deep Reinforcement Learning to solve the reconfiguration problem in the context of experience-driven networking. It has also

been used for QoE estimation of video streaming [35], [36]. The transfer learning technique we use (CORAL [26]) has already been used in optical networks for assisted quality of transmission estimation of an optical lightpath [37]. Here, we apply it to the RTC scenario, trying to align statistical features of network traffic.

Focusing on RTC traffic, many works propose techniques to identify it among other traffic categories. The authors of [38] use a stochastic characterization of Skype traffic to obtain an ML-based model to be used for classification. In [39], UDP flows are classified into different classes, including Skype and RTP-based traffic,using SVM models and statistical signatures of the payload. The approach proposed in [40] leverages statistical properties of RTP to differentiate between voice and data traffic. The authors of [41] propose a method to detect WebRTC sessions at run-time based on statistical pattern recognition. Finally, some approaches target signaling mechanisms of RTC applications to identify Skype traffic through in-clear headers exchanged during session setup [42]. The ultimate goal of RTC traffic classification is the improvement of QoS and users' QoE. These aspects have been studied, focusing on the relationship between QoS and QoE [43], targeting the WebRTC [44] and mobile [45] scenarios. Another way to improve the QoE of RTC traffic is optimal media bridge placement. The media bridge relays the traffic between the peers. Some works target cache placement in SDN, which can be adapted to the RTC scenario [46], [47], [48], [49].

Fewer works address the classification of media streams carried by RTP streams. Authors of [50] train machine learning classifiers to distinguish, among other classes, video and audio flows, targeting the WeChat messaging application. The approach presented in [51] identifies 20 codecs used for compression of audio,based on packet size, RTP timestamp delta, payload type and ratio between RTP timestamp delta and payload size. However, they do not use machine learning but a simple lookup table. In [52], the authors use statistics on the packet size as a distinguishing feature between audio, video streaming, browser, and chat traffic. They use interesting features, albeit fewer than we do, and divide into broader traffic classes. We only target RTC traffic and divide it into 7 classes, while for them it is a single macro class. As a model, they opt for an interpretable decision tree, similar to ours.

The closest work to ours is the approach proposed by Choudhury *et al.* [53]. There, the authors design a system to classify RTP traffic to the employed codec. They develop an ML pipeline similar to ours, to classify audio traffic into three Variable Bit Rate (VBR) codecs, thus identifying three types of audio. Conversely, we distinguish seven classes, two of which are audio (audio and FEC audio), four are video (three video qualities and FEC video) and one is screen sharing. Similar to us, they classify RTP streams separately by time bin, with a granularity coarser than ours – 10-20 seconds vs 1 second. They use two types of features: statistical features of packet sizes (such as mean, standard deviation, mode, etc.) and entropy-based features (4 types of entropy calculations on the RTP payload of the packets). We follow a similar approach, using five feature groups and calculating various statistics on the distributions. They also perform feature selection, reducing from 10 to 7 features. We use 8 for Webex and 4 for Jitsi. Like in our system, they train offline, using 18-second streams and then the classifier is deployed in real time, over 10 seconds of stream data. They get overall 97% accuracy, similar to us (95%). Concerning the algorithms, they opt for a 1 Nearest Neighbours, while we finally choose a Decision Tree.

In summary, our work aims at unveiling the nature of media streams. Differently from previous works, we classify streams into a rich set of classes including media type (audio and video), video quality and redundant data (FEC). We engineer a wider range of features and then run an thorough feature selection process. Moreover, to the best of our knowledge, we are the first ones to explicitly target real-time applications with a 1 second (or shorter) classification delay, while the past approaches base their decision on the characteristics of an entire stream, lasting 10 seconds or longer.

VIII. Conclusion and Future Work

In this paper, we proposed a machine learning approach to classify the media streams generated by RTC applications in real time. Given a media stream carried within the RTP protocol, we can distinguish seven different classes, including different video qualities, screen sharing and redundant data used to mitigate losses (i.e., the FEC). We carefully engineered features based on packet characteristics and designed the system to work with a minimal set of features using a light yet accurate tree-based model. We chose Webex Teams and Jitsi Meet as case studies and showed that we achieve high classification performance with only 1s classification delay. Our approach is robust to the choice of classification algorithm and rarely commits systematic errors. Our experiments show that it is possible to use a model trained for one application to classify streams of another, albeit with a performance penalty. If non-labelled data from the target application are available, it is possible to use transfer learning techniques to achieve better results.

Our approach is designed as a building block of a network management system that optimizes traffic engineering for RTC applications. Our future work goes in this direction, and our approach enables the network control plane to make decisions on traffic with the awareness of RTC traffic. Our final goal is to measure and optimize the QoE perceived by users of RTC applications. We publish our code and dataset to encourage research in this direction.

Acknowledgements

This work has been supported by the SmartData@PoliTO center on Big Data and Data Science and Cisco Systems Inc.

References

[1] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five years at the edge: Watching internet from the isp network," *IEEE/ACM Trans. on Networking*, vol. 28, no. 2, pp. 561–574, 2020.

[2] R. Frederick, S. L. Casner, V. Jacobson, and H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications." RFC 1889, Jan. 1996.

[3] A. Nistico, D. Markudova, M. Trevisan, M. Meo, and G. Carofiglio, "A comparative study of RTC applications," in *2020 IEEE International Symposium on Multimedia (ISM)*, pp. 1–8, IEEE, 2020.

[4] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the" s" in https," in *Proc. of the 10th ACM International on Conf. on emerging Networking Experiments and Technologies*, pp. 133–140, 2014.

[5] M. Finsterbusch, C. Richter, E. Rocha, J. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.

[6] G. Perna, D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafò, and G. Carofiglio, "Online classification of rtc traffic," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–6, IEEE, 2021.

[7] K. Norrman, D. McGrew, M. Naslund, E. Carrara, and M. Baugher, "The Secure Real-time Transport Protocol (SRTP)." RFC 3711, Mar. 2004.

[8] J. Rosenberg, C. Huitema, R. Mahy, and J. Weinberger, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)." RFC 3489, Mar. 2003.

[9] P. Matthews, J. Rosenberg, and R. Mahy, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)." RFC 5766, Apr. 2010.

[10] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols." RFC 5245, Apr. 2010.

[11] M. Petit-Huguenin and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)." RFC 7983, Sept. 2016.

[12] C. Holmberg, S. Hakansson, and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements." RFC 7478, Mar. 2015.

[13] M. Westerlund and S. Wenger, "RTP Topologies." RFC 7667, Nov. 2015.

[14] G. Carofiglio, G. Grassi, E. Loparco, L. Muscariello, M. Papalini, and J. Samain, "Characterizing the relationship between application qoe and network qos for real-time services," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, pp. 20–25, 2021.

[15] R. C. Streijl, S. Winkler, and D. S. Hands, "Mean opinion score (mos) revisited: methods and applications, limitations and alternatives," *Multimedia Systems*, vol. 22, no. 2, pp. 213–227, 2016.

[16] D. Vucic and L. Skorin-Kapov, "The impact of packet loss and google congestion control on qoe for webrtc-based mobile multiparty audiovisual telemeetings," in *International Conference on Multimedia Modeling*, pp. 459–470, Springer, 2019.

[17] International Telecommunication Union – Telecommunication Standardization Bureau, "Recommendation ITU-T G.1070 – Opinion model for video-telephony applications," 2018.

[18] International Telecommunication Union – Telecommunication Standardization Bureau, "Recommendation ITU-T G.107.1 – Wideband E-model," 2019.

[19] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi, "Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 163–169, 2017.

[20] L. Deri and N. SpA, "nprobe: an open source netflow probe for gigabit networks," in *TERENA Networking Conference*, pp. 1–4, 2003.

[21] D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafo, and G. Carofiglio, "What's my App?: ML-based classification of RTC applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 48, no. 4, pp. 41–44, 2021.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[24] L. Tolоşi and T. Lengauer, "Classification with correlated features: unreliability of feature ranking and solutions," *Bioinformatics*, vol. 27, no. 14, pp. 1986–1994, 2011.

[25] L. Y. Pratt, "Discriminability-based transfer between neural networks," *Advances in neural information processing systems*, pp. 204–204, 1993.

[26] B. Sun, J. Feng, and K. Saenko, "Correlation alignment for unsupervised domain adaptation," in *Domain Adaptation in Computer Vision Applications*, pp. 153–171, Springer, 2017.

[27] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[28] M. Trevisan, I. Drago, M. Mellia, H. H. Song, and M. Baldi, "What: A big data approach for accounting of modern web services," in *2016 IEEE Int. Conf. on Big Data (Big Data)*, pp. 2740–2745, IEEE, 2016.

[29] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning," in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–8, IEEE, 2018.

[30] A. Balachandran, V. Aggarwal, E. Halepovic, J. Pang, S. Seshan, S. Venkataraman, and H. Yan, "Modeling web quality-of-experience on cellular networks," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, pp. 213–224, 2014.

[31] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A machine learning approach to classifying youtube qoe based on encrypted network traffic," *Multimedia tools and applications*, vol. 76, no. 21, pp. 22267–22301, 2017.

[32] P. Casas, A. D'Alconzo, F. Wamser, M. Seufert, B. Gardlo, A. Schwind, P. Tran-Gia, and R. Schatz, "Predicting qoe in cellular networks using machine learning and in-smartphone measurements," in *Ninth International Conf. on Quality of Multimedia Experience*, pp. 1–6, IEEE, 2017.

[33] E. Baştuğ, M. Bennis, and M. Debbah, "A transfer learning approach for cache-enabled wireless networks," in *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pp. 161–166, 2015.

[34] Z. Xu, D. Yang, J. Tang, Y. Tang, T. Yuan, Y. Wang, and G. Xue, "An actor-critic-based transfer learning framework for experience-driven networking," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 360–371, 2021.

[35] S. Ickin, M. Fiedler, and K. Vandikas, "Customized video qoe estimation with algorithm-agnostic transfer learning," *arXiv preprint arXiv:2003.08730*, 2020.

[36] Y. Hao, J. Yang, M. Chen, M. S. Hossain, and M. F. Alhamid, "Emotion-aware video qoe assessment via transfer learning," *IEEE MultiMedia*, vol. 26, no. 1, pp. 31–40, 2019.

[37] C. Rottondi, R. di Marino, M. Nava, A. Giusti, and A. Bianco, "On the benefits of domain adaptation techniques for quality of transmission estimation in optical networks," *Journal of Optical Communications and Networking*, vol. 13, no. 1, pp. A34–A43, 2021.

[38] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: when randomness plays with you," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 37–48, 2007.

[39] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection classifier for udp traffic," *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1505–1515, 2010.

[40] A. S. Buyukkayhan, A. Kavak, and E. Yaprak, "Differentiating voice and data traffic using statistical properties," in *2013 International Conference on Electronics, Computer and Computation (ICECCO)*, pp. 76–79, 2013.

[41] M. Di Mauro and M. Longo, "Revealing encrypted webrtc traffic via machine learning tools," in *2015 12th International Joint Conference on e-Business and Telecommunications*, vol. 04, pp. 259–266, 2015.

[42] T. Sinam, I. T. Singh, P. Lamabam, N. N. Devi, and S. Nandi, "A technique for classification of voip flows in udp media streams using voip signalling traffic," in *2014 IEEE International Advance Computing Conference (IACC)*, pp. 354–359, 2014.

[43] N. Rao, A. Maleki, F. Chen, W. Chen, C. Zhang, N. Kaur, and A. Haque, "Analysis of the effect of qos on video conferencing qoe," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 1267–1272, IEEE, 2019.

[44] B. Garcia, M. Gallego, F. Gortazar, and A. Bertolino, "Understanding and estimating quality of experience in webrtc applications," *Computing*, vol. 101, no. 11, pp. 1585–1607, 2019.

[45] M. Vaser and S. Forconi, "Qos kpi and qoe kqi relationship for lte video streaming and volte services," in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pp. 318–323, IEEE, 2015.

[46] J. Badshah, M. Mohaia Alhaisoni, N. Shah, and M. Kamran, "Cache servers placement based on important switches for sdn-based icn," *Electronics*, vol. 9, no. 1, p. 39, 2020.

[47] J. Badshah, M. Kamran, N. Shah, and S. A. Abid, "An improved method to deploy cache servers in software defined network-based information centric networking for big data," *Journal of Grid Computing*, vol. 17, no. 2, pp. 255–277, 2019.

[48] D. Kim and Y. Kim, "Enhancing ndn feasibility via dedicated routing and caching," *Computer networks*, vol. 126, pp. 218–228, 2017.

[49] S. Clayman, R. S. Kalan, and M. Sayit, "Virtualized cache placement in an sdn/nfv assisted sand architecture," in *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pp. 1–5, IEEE, 2018.

[50] M. Shafiq, X. Yu, and A. A. Laghari, "Wechat traffic classification using machine learning algorithms and comparative analysis of datasets,"

International Journal of Information and Computer Security, vol. 10, no. 2-3, pp. 109–128, 2018.

[51] P. Matousek, O. Rysavy, and M. Kmet, "Fast rtp detection and codecs classification in internet traffic," *Journal of Digital Forensics, Security and Law*, 01 2014.

[52] M. C. S, S. H, and T. E. Somu, "Network traffic classification by packet length signature extraction," in *2019 IEEE International WIE Conference on Electrical and Computer Engineering*, pp. 1–4, 2019.

[53] P. Choudhury, K. R. Prasanna Kumar, G. Athithan, and S. Nandi, "Analysis of vbr coded voip for traffic classification," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 90–95, 2013.



Gianluca Perna is a PhD student at Politecnico di Torino and member of SmartData@Polito research center for Big Data technologies. He obtained a Bachelor's degree in Telecommunication engineering and a Master's degree in ICT For Smart Societies with an excellent grade, both in the same University. Thanks to his passion and expertise in Machine Learning, Internet of Things and Data analysis he obtained a six months research grant before starting his PhD in the SmartData group. He is also pursuing a patent in the field of Building Design and participating in an international project with the leading IT company Cisco Systems.



Dena Markudova is a PhD student in Electrical, Electronics and Communications Engineering at Politecnico di Torino, Italy and member of the SmartData@Polito research center. Her research focuses on Data science applied to Computer Networking – traffic analysis and application of Machine learning algorithms for better Network management. She obtained her Bachelor degree in Telecommunications at "Ss. Cyril and Methodius" University in Skopje, North Macedonia in 2016 and her Master's degree in ICT for Smart Societies at Politecnico di Torino in 2018.



Martino Trevisan received his PhD in 2019 from Politecnico di Torino, Italy. He is currently an assistant professor with the University of Trieste. He has been collaborating in both Industry and European projects and spent six months in Telecom ParisTech, France working on High-Speed Traffic Monitoring during his M.Sc. He visited the Cisco labs in San Jose twice, in the summers of 2016 and 2017, as well as AT&T labs during fall 2018. He was also a Visiting Professor at the Federal University of Minas Gerais in Brazil in 2019.



Paolo Garza received the master's and PhD degrees in computer engineering from the Politecnico di Torino. He has been an associate professor at the Dipartimento di Automatica e Informatica, Politecnico di Torino, since December 2018. He spent three years as an assistant professor at Politecnico di Milano. He coauthored about 100 papers in the areas of data mining and machine learning. His current research interests are in the fields of data mining, database systems, and big data analytics. He has worked on classification, clustering, itemset mining and scalable algorithms.



Michela Meo is a Professor of Telecommunication Engineering with the Politecnico di Torino. She coauthored about 200 papers, 80 of which on international journals. She edited a book Green Communications (Wiley) and several special issues of international journals. Her research interests include green networking, energy-efficient mobile networks and data centers, Internet traffic classification, and characterization. Prof. Meo was an Associate Editor of ACM/IEEE Transactions of Networking, Green Series of the IEEE Journal on Selected Areas of Communications Networking and IEEE Communication Surveys and Tutorials. She is a Senior Editor of IEEE Transactions on Green Communications. In the role of a General or Technical Chair, she has led the organization of several conferences, including ITC, ICC symposia, ISCC. She chairs the International Advisory Council of the International Teletraffic Congress. She was the Deputy Rector of Politecnico di Torino from March 2017 to March 2018.



Maurizio Matteo Munafò is Assistant Professor at the Department of Electronics and Telecommunications of Politecnico di Torino. He holds a Dr.Ing. degree in Electronic Engineering since 1991 and a Ph.D. in Telecommunications Engineering since 1994, both from Politecnico di Torino. He has coauthored about 80 journal and conference papers in the area of communication networks and systems. His current research interests are in simulation and performance analysis of communication systems and traffic modeling, measurement, and classification.



Giovanna Carofiglio received the Dr Ing degree in telecommunication engineering and in electronic and telecommunication engineering, both from Politecnico di Torino, Italy, in 2004, and the PhD in telecommunication engineering jointly from Politecnico di Torino and Telecom ParisTech, Paris, France, in 2008. Her graduate reseach focused on stochastic analysis of wired and wireless networks and has been performed at Politecnico di Torino and at Ecole Normale Superieure (ENS Ulm) in the INRIA-TREC group. She spent more than six years at Bell Labs, as head of the research department on content networking. She currently works at Cisco Systems as a Distinguished Engineer. She was general co-chair of ACM ICN 2014. She is a member of the IEEE.