Machine-Learning Based Prediction of Next HTTP Request Arrival Time in Adaptive Video Streaming

Andrea Pimpinella[1], Alessandro E. C. Redondi[1], Frank Loh[2], Michael Seufert[2]
[1]DEIB - Politecnico di Milano, Milan, Italy, [2]University of Würzburg, Würzburg, Germany
{andrea.pimpinella,alessandroenrico.redondi}@polimi.it,
{frank.loh, michael.seufert}@uni-wuerzburg.de

***Abstract*—Continuously monitoring the network activity to proactively recognise possible problems and prevent users QoE degradation is a major concern for network operators, for both mobile radio and home networks. Considering video streaming applications, which generate the majority of overall Internet traffic, monitoring the chunk requests from the video client to the video server is of particular interest, as they not only indicate that a download burst is imminent, but their type (e.g., request of an audio or video chunk) and frequency also allow to estimate which and how much data will be downloaded to the client. In this work, we propose a machine-learning based video streaming traffic monitoring architecture able to i) predict when next uplink request will be issued by the video client and ii) classify the type of next uplink request. We evaluate the system performance on a dataset of more than 900 HTTP adaptive streaming sessions and 15,000 request-response exchanges, where both the predictor of the next request arrival and the request type classifier are fed with lightweight features extracted from encrypted traffic in an online fashion, both in the uplink and downlink directions of the traffic. Results show that i) the system is able to classify the type of a HAS uplink requests with an accuracy greater than 95 % and ii) pipe-lining request type classification and prediction of next request arrival time improves the final prediction performance.**

***Index Terms*—Network monitoring, HTTP adaptive video streaming, machine learning, encrypted traffic.**

I. Introduction

The recent improvements in networking technologies, end devices, and digital services are boosting the generation of Internet traffic worldwide. It is well known that the majority of such traffic carries video streaming contents: as an example, in mobile networks more than 50 % of the downlink traffic is due to video streaming applications, according to a recent Sandvine report [1]. Similarly, in home networks a shift towards higher video resolution like UHD or 4K is clearly visible [2].

From the perspective of an Internet service provider or a mobile network operator, dealing with high volumes of video streaming traffic is particularly challenging. Indeed, such type of traffic constitutes the largest part of total traffic volume, therefore driving many of the processes related to network management and optimization. At the same time, video streaming comes with tight QoE requirements, which if not met can increase users dissatisfaction and consequently the churn rate to other operators.

Network operators generally take smart management approaches and use resource optimization techniques to guarantee QoE requirements in an efficient way. Generally, such techniques are based on the knowledge of aggregated spatio-temporal traffic patterns, which are known to be recurrent in large scale networks [3], [4]. However, traffic peaks and fluctuations from the normal behaviours can greatly affect the performance of such techniques.

Rather than relying on long-term traffic patterns for managing the resources, a different approach consists in monitoring continuously the network activity, trying to anticipate possible problems and quickly react if a QoE degradation is predicted. As an example, considering video streaming traffic, monitoring the chunk requests from the video client to the video server is of particular interest, as they not only indicate that a download burst is imminent, but their type (e.g., request of an audio or video chunk) and frequency also allow to estimate which and how much data will be downloaded to the client.

The approach comes with its own challenges: first, the operator generally does not have access to the payload of the packets flowing in the network due to the increasing use of application-layer encryption (HTTPS). Therefore, any monitoring approach of this type must rely only on network traffic characteristics. Second, if monitoring is applied on the traffic produced by each user separately, rather than on traffic aggregates, it must be based on extremely lightweight processing techniques.

With this paper, we propose a monitoring architecture that meets such requirements. Our contributions are as follows:

1) We focus on HTTP Adaptive Streaming (HAS) sessions, and we propose a methodology to predict the next HAS uplink request arrival based on a dataset of more than 15,000 requests and 900 YouTube streaming sessions.
2) Since HAS requests may be of different types (audio/video), whose knowledge is of paramount importance to characterize future traffic behaviour, we also propose a classifier able to differentiate between them with excellent performance.
3) Both, the predictor of the next request arrival and the request type classifier are fed with lightweight features extracted from encrypted traffic in an online fashion.
4) We evaluate the monitoring architecture with extensive experiments, considering features extracted solely in the uplink direction or in both directions of the traffic.

The remainder of this paper is structured as it follows: Section **??** summarises related works, while Section **??** introduces the dataset and the monitoring architecture. Section III

comments on the performance obtained from uplink requests type classification and next arrival time prediction. Finally, Section IV concludes and outlines future research directions.

II. Related Works

Especially in the context of network management, network traffic classification prediction is important for many application areas. An early survey for network traffic analysis and traffic prediction techniques is given by Joshi in [5], and especially for real-time traffic by Barros in [6].

For what concerns video streaming, more specific works are present for video flow detection and classification [7], video and audio flow separation [8], and overall QoE prediction with either modeling [9] or machine learning [10], [11], [12] techniques. In addition, a transfer learning approach across different networks is presented by Orsolic in [13]. There, the authors use two large YouTube datasets and the influence of different locations on the model performance. Furthermore, the recent question in literature is if video QoE prediction is preferable from full network traces like done in e.g., [14] or if uplink requests are enough, as presented by Gutterman [15]. For video QoE prediction based on uplink requests, knowledge about the arrival time and size of upcoming requests, and the overall uplink throughput is essential. While a deep learning based uplink throughput prediction framework is presented by Lee [16], a more detailed uplink request arrival prediction is, to the best of our knowledge, not tackled in literature so far.

A. Background

Video streaming services (YouTube, Netflix, etc.) make use of (encrypted) HAS streaming techniques for delivering content (e.g., MPEG-DASH). According to such techniques, the content on the server is divided into a sequence of segments, containing either audio or video content. Each segment contains just a short interval of the original content and is available at different bit-rates. A client willing to stream a video issues HTTP requests to the server, selecting which segment to retrieve and at which bit-rate. Generally, the client requests the maximum bit-rate that can be downloaded in time without causing stalls or re-buffering events. Sometimes, the client opens multiple connections to the server (either TCP or QUIC) in order to parallelize requests. Figure 1 illustrates a typical HAS request-response exchange: first, the client issues an HTTP request (dashed line) h_i to the server, which after δ_i seconds replies with a burst of n packets (blue lines) containing the requested segment. The burst lasts for r_i seconds, depending on the requested segment bit-rate. After τ_i seconds from the last packet of the burst, the client issues another request h_{i+1}. The goal of this work is twofold: based solely on low-cost features extracted by the i-th client-server exchange illustrated in Figure 1, we aim to (i) classify the type of HTTP request transmitted by the client (i.e., audio or video[1]) and (ii) predict when the next request h_{i+1} will be issued (i.e., provide an estimate for the interval $\delta_i + r_i + \tau_i$).

[1]Requests for mixed contents are not considered in this work.

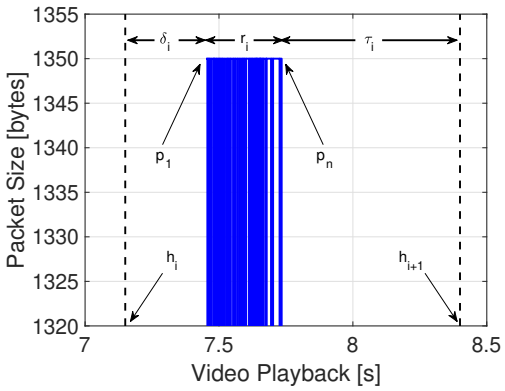

Fig. 1. Server response R_i to the i-th HAS client request, issued at time h_i = 7.15 s during the playback of a stream taken as example. In blue, the burst received from the server, composed of n packets sized 1350 bytes each.

B. The Dataset

For the task at hand, we used a dataset containing more than 15,000 YouTube video sessions, streamed and recorded from June 2018 to February 2019. The dataset was created with a monitoring tool similar to [17]. The video sessions were streamed from different networks (home WiFi network, corporate WiFi network, LTE mobile network), from four different geographic locations (Austria, France, Germany, Italy), and from four different ISPs. During the streaming, we additionally operated a network emulator, which could artificially limit the bandwidth on both uplink and downlink.

The measurement framework itself ran on a laptop device and utilized the Selenium browser automation library. For each measurement run, it started a Chrome browser and browsed to a randomly selected YouTube video, where the streaming of that video was automatically started. Chrome was configured such that QUIC traffic was enabled and all HTTP requests were logged during the streaming session. A JavaScript-based monitoring script [18], [19] was injected into the web page to periodically record every 250 ms the current timestamp, as well as the current video playtime, buffered playtime, video resolution, and player state. During the streaming session, the presence of pre-roll or mid-roll advertisement clips was monitored, and several user interactions could be emulated, such as pausing, scrubbing to a different playback position, or changing the video resolution. After 180 s, the streaming session was aborted by closing the browser.

During the whole streaming session, we captured the (encrypted) network traffic using tshark to log basic packet information (timestamp, source IP, source port, destination IP, destination port, size), as well as DNS lookup responses to obtain a mapping between IP addresses and domain names. In each network trace, we identified TCP and QUIC YouTube video flows based on the domain name (googlevideo.com), and only considered those YouTube video flows for our analysis, i.e., we ignored all other flows.

The characteristics of the full dataset are described in detail in [11]. To obtain a homogeneous dataset for our initial

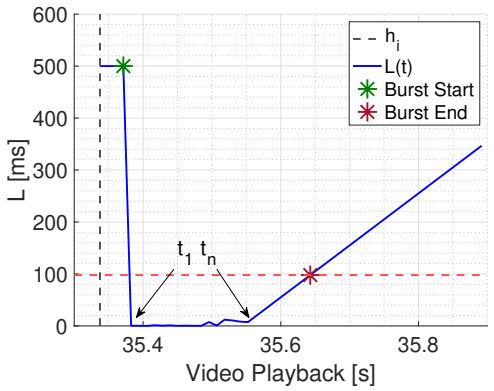

Fig. 2. Server burst detection procedure.

analysis, in this work, we selected only those video streaming sessions without any imposed network emulation, without any emulated user interactions, and without any advertisement clips. This comprises a subset of 900 YouTube video sessions and more than 16,000 HAS uplink requests. The maximum observed bandwidth in these sessions was roughly 20 Mbps.

C. Data Preprocessing

The dataset is pre-processed in the following way. First, for each video session, we exclude from the analysis the first 15 seconds of playback time, due to the noisy behaviour of the initial buffering phase. Then, considering that requests from the client can be carried by multiple QUIC and TCP traffic flows, we restrict our analysis to the dominant flow of each video session, i.e., the TCP or QUIC flow which carries the majority of the video session traffic. According to this filtering operation, we end up with roughly 15,000 HTTP request-response exchanges, 60 % of the times carried by a TCP flow and 40 % of the times by a QUIC one. Finally, we label each uplink request with the corresponding type, i.e., either audio or video. With this aim, we pre-process the log file returned by the JavaScript-based monitoring script, retaining only the entries which correspond to HTTP request instances. Then, for each request, we identify the corresponding *itag*, a parameter carrying information on the type of the request (audio/video) as well as the requested quality. At the end of this procedure, we recognise roughly 10,000 requests for video contents while only 5,000 of them are for audio contents. Also, we observe that when QUIC traffic is enabled 45 % of the requests are for audio contents (i.e., 55 % of the time a request carried by QUIC is for a video content) whereas this happens only 32 % of the times when TCP is used (i.e., 68 % of the requests are for video contents in this case).

D. Online Feature Extraction

Next, each video session is processed in order to extract characteristic features as well as ground truth information useful for training supervised machine learning models. The general process is illustrated in the left-side of Figure 3. In particular, we opt for processing techniques which can be implemented in an online fashion, i.e., as the network traffic flows. We recognise three main steps, namely client request detection, response burst detection and features extraction.

1) Client request detection: We observe that, in a video session, client requests are the only uplink packets greater than 100 bytes (all the rest are essentially acknowledgments), regardless of the request type (audio or video) and transport layer protocol used (TCP or QUIC). A simple threshold on the dimension of uplink packets is therefore used to identify a client request, together with its timestamp h_i.

2) Response burst detection: Upon the detection of a new request h_i, we move our attention to downlink traffic in order to highlight the response burst R_i coming from the server. For the task at hand, we start a timer $L(t)$ with a timeout of 100 ms after the detection of the request. The timer is reset every time a downlink packet larger than 250 bytes is received from the server. A burst is detected when the timeout elapses and at least $n = 50$ packets have been observed. Note that both the timeout, the minimum packet size, and the minimum burst length are set empirically after a trial and error procedure over different configurations. We show in Figure 2 an example of such procedure: in this case, a new request h_i is detected at time 35.34 s, after roughly 500 ms from last downlink packet. Then, the first packet p_1 of server response R_i is received at time $t_1 = 35.37$ s, for a total burst length of 202.8 ms. Note that the end of the burst is detected at time $t_n + 100$ ms $= 35.64$ ms, i.e., 100 ms after last packet of R_i is received. Finally, we set $\delta_i = t_1 - h_i$ and $r_i = t_n - t_1$.

3) Features extraction: The detection of client requests and server responses allows to extract two sets of features, relative to the uplink and downlink direction of traffic:

- Client Request (**CR**) features: they model the sole activity of the client when sending the i-th HTTP request, without considering the response from the server. Therefore, they can be computed right after the client request is detected.
- Server Response (**SR**) features: they provide details about R_i, i.e., the server's response to the i-th HTTP request. Therefore, their computation must wait the end of the burst detection process.

For what concerns CR features, the following quantities are computed after the detection of h_i:

- *Inter-Request Time*: the difference between the detected request and the previous one, i.e., $h_i - h_{i-1}$.
- *Request Size*: The size in bytes of the HTTP request.
- *Request Sequence Number and Playback Duration*: a general behavior of video streaming clients is to produce many consecutive requests at the beginning of the streaming session to fill the video buffer as much as possible. Therefore, we keep track of the request sequence number i as well as the time elapsed since the first request.
- *Moving Averages*: it has been shown [11], [20] that adjacent requests are not uncorrelated, and that information about recent history of the streaming session can give information on its future behavior. For this reason, we compute the moving average of both the inter-request

TABLE I
FEATURES RELATED TO THE i-TH EXCHANGE OF A VIDEO HAS SESSION. MOVING AVERAGES FEATURES ARE NOT SHOWN FOR SPACE LIMITS.

Set	Feature	Description
CR	$h_i - h_{i-1}$	Time lag between the i-th HTTP request and the previous one
	i	Sequence Number of i-th HTTP request
	Request Size	Size of the i-th request [B]
	Playback Duration	Time length of the already played-back content up to h_i
SR	δ_i	Inter request-response delay
	Response Size	Size of R_i [B]
	n	Length of R_i [N. of Packets]
	r_i	Download Time
	τ_{i-1}	Time between last packet of R_{i-1} received by the client and h_i
	Inter-Packets Arrival Time distribution	25-th, 50-th and 75-th percentiles of IPA Time
	Inter-Packets Arrival Time Coefficient of Variation	Ratio between standard deviation and mean of IPA Time

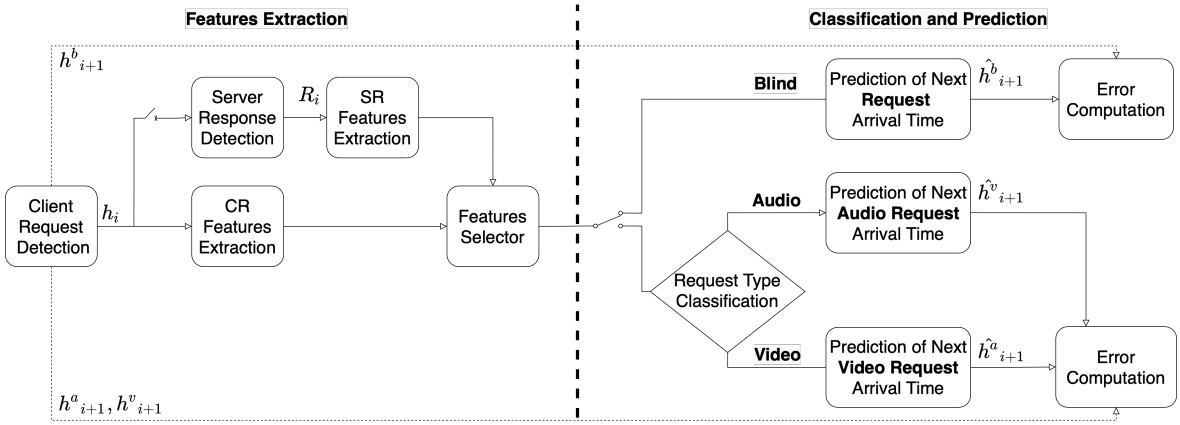

Fig. 3. Architecture for video streaming traffic monitoring and prediction of next uplink request arrival time. On the left side, the features extraction process is represented: the trade-off here consists to compute either uplink features only or add also downlink features (i.e., wait for server response). On the right side, the two prediction approaches are represented, either Blind (upper branch) or Audio/Video (lower branch). According to the latter approach, we first classify the type of last observed request and then we predict the arrival of next request of the same type.

time and the request size, using a window of $N = 10$ requests, where N is set empirically among different tested options. For the request size, we also compute the moving standard deviation.

As for SR features, we compute the following quantities:

- *Inter Response-Request Time*: the time elapsed between the last packet of the last server response and the new client request, i.e., τ_{i-1}.
- *Request-Response Delay*: the time elapsed between the client request and the first packet of the server response, i.e., δ_i.
- *Response Size*: the total volume in bytes of the response, as well as the number of packets n constituting the response burst.
- *Download Time*: the time elapsed from the first to the last packet of the response burst, i.e., r_i.
- *Inter-Packet Arrival Time distribution*: we also compute the inter arrival times of the packets in the response burst and estimate the 25-th, 50-th and 75-th percentiles of their distribution.
- *Inter-Packet Arrival (IPA) Time Coefficient of Variation*: the ratio between the standard deviation and the mean of the inter arrival times of the packets in the response burst.
- *Received Volume*: we also keep track of the cumulative data volume received by the client as well as the total received number of packets.
- *Moving Averages*: similarly to the CR case, we compute moving averages of the response size, the response download time and the inter request-response delay, as well as their standard deviations.

Note that both CR and SR features, summarized in Table I, can be easily updated for each new request issued by the streaming client, making them suitable for online analysis[2].

III. EXPERIMENTS AND RESULTS

In this section we use the features introduced in Section II-D to i) classify HAS uplink requests and ii) predict when next request is transmitted. The former task is a binary classification problem, as client requests can be either for audio or video content. Differently, the latter is a regression problem, which we tackle distinguishing two alternative approaches. The first approach is represented in the upper branch of the right side of Figure 3. According to this approach, when a new uplink request is observed features are computed and the arrival time of next request is predicted. Thus, in this case we do

[2]In this case, the only overhead is due to the packets live capturing system. While this aspect is not discussed in this work, it can be shown [11] that its impact on performance is negligible for lightweight prediction models.

not detect the type of the last observed request and we are blinded about the type of next request as well. For these reasons, we refer to this approach as *Blind* and we refer to the corresponding prediction output as $\hat{h^b}_{i+1}$. Conversely, in the second approach, we first classify the last observed request as being an audio or video request and consequently we predict the arrival time of the next request of the same type, i.e., either $\hat{h^a}_{i+1}$ or $\hat{h^v}_{i+1}$ respectively. As represented in the lower branch of the right side of Figure 3, this lets us customise the features to the specific request type and thus set up two different regression algorithms, one for audio and one for video requests. We will refer to this approach as either *Audio* or *Video*, according to the request type. In the next sections we present the performance obtained for both the classification (III-A) and the regression (III-B) tasks.

A. Classification of HAS Uplink Requests

A common way to solve binary classification problems is by supervised learning: among the several available machine-learning classifiers we select the Random Forest (RF) algorithm, which is widely known to perform well in general and has been successfully applied in the past for similar problems [11], [12], [10], [14]. Operatively, the set of streaming sessions is divided in training ($\mathcal{X}$) and test ($\mathcal{Y}$) sets, with splitting ratios of 0.9 and 0.1 according to a 10-fold cross validation strategy. Secondly, we use $\mathcal{X}$ to select the best set of hyper-parameters (number of decision trees, number of features to be considered in each tree, etc.) of the RF algorithm. Finally, we train the RF with the best hyper-parameters found and we test its performance on the test set. Figure 4 shows the confusion matrices to summarise the classification performance. We consider two cases: in the former, only CR features are used, whereas in the latter we leverage also SR features (that is, the client request is classified after the server response has been observed). As one can see in the left-side of Figure 4, when only uplink features are considered the overall accuracy equals 79.3 %, where 40.7 % is due to correct classification of audio requests while 38.6 % is associated to video requests. However, as shown in the right-side of Figure 4, when both CR and SR features are used for classification the performance improve by 7.6 % and 8.6 % for audio and video requests respectively, raising the overall accuracy to 95.5 %. To explain the benefit observed by including SR features in the classification process, we show in Figure 5 the scatter plot of the response download time versus the response size with respect to the uplink requests in the test set, when either QUIC (left) or TCP (right) is adopted. As one can see, most of audio contents are shorter than 1000 KB and 750 KB when QUIC or TCP protocols are used, whereas this happens only for 75 % and 80 % of video contents respectively. Considering that in our experiments the maximum observed bandwidth is roughly 20 Mbps, this leads on average to 0.5 s shorter download times for audio contents with respect to video ones. Similar observations can be drawn observing the scatter plots of other SR features couples, which are not shown for space limits.

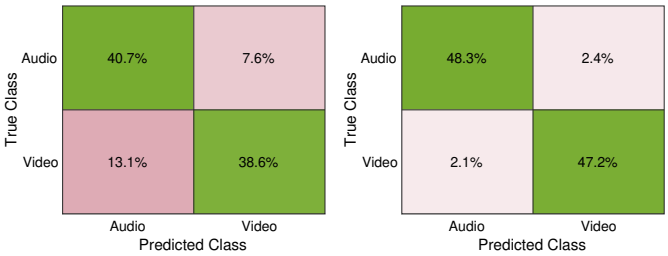

Fig. 4. Confusion plots for HAS uplink request type classification, when either UL (left) or UL+DL (right) model is selected.

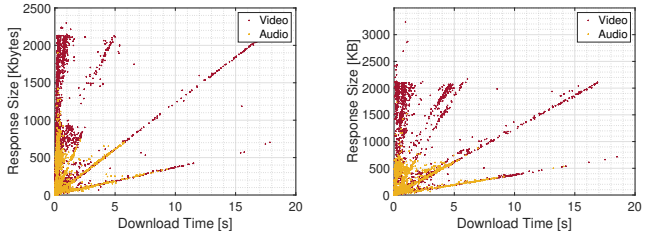

Fig. 5. Download Time vs Burst Size for audio (yellow) and video (red) flows, when either QUIC (left) or TCP (right) protocol is used.

B. Prediction of Next HAS Uplink Request Arrival Time

For what concerns the prediction of the next uplink request arrival time, regardless of the selected approach (i.e., either Blind, Audio, or Video), we consider several prediction models characterised by different complexity and features sets. For what concerns complexity, we focus on baseline models, obtained by either constant prediction or simple data copy, and RF models, where a set of features is fed into a RF regressor to output the target. For what concerns the feature sets, we distinguish a first case where only uplink traffic related features are considered and a second case where also downlink traffic related features are included for prediction. Using the same training and test set splits of Section III-A, we consider the following prediction models:

- Static Baseline (**SB**): this baseline model estimates the arrival time of request h_{i+1} as constantly equal to:
$$\hat{h_{i+1}} = \frac{\sum_{s \in \mathcal{X}, \forall j} (h^s{}_j - h^s{}_{j+1})}{|\mathcal{X}|} \quad (1)$$
i.e., the average time between two consecutive requests computed over all the instances j of each stream s $\in \mathcal{X}$.
- Dynamic Baseline (**DB**): this baseline model predicts h_{i+1} as:
$$\hat{h_{i+1}} = h_i + (h_i - h_{i-1}) \quad (2)$$
i.e., it copies the last observed inter-request time.
- Uplink (**UL**): this model leverages only CR features and predicts h_{i+1} as equal to:
$$\hat{h_{i+1}} = h_i + f(CR) \quad (3)$$
where f(·) represents the RF regressor and its output equals $\hat{\delta}_i + \hat{r}_i + \hat{\tau}_i$.

TABLE II
FREQUENCIES OF UPLINK REQUESTS WHEN EITHER BLIND, AUDIO OR VIDEO APPROACH IS SELECTED, FOR QUIC AND TCP TRAFFIC FLOWS.

	QUIC [s]	TCP [s]
Blind	5.12	5.17
Audio	9.71	12.27
Video	10.75	9.15

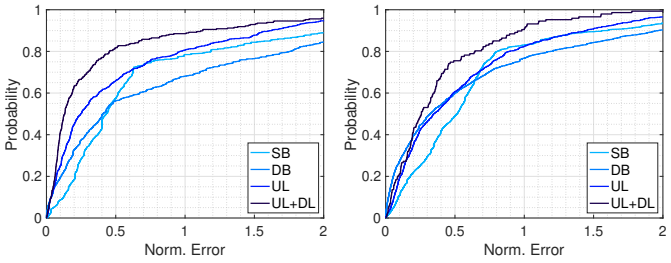

Fig. 6. Blind, QUIC (left) and TCP (right) traffic flows.

- Uplink + Downlink (**UL+DL**): this model leverages both CR and SR features and predicts h_{i+1} as:

$$\hat{h_{i+1}} = h_i + \delta_i + r_i + f(CR, SR) \quad (4)$$

where f(CR, SR) equals $\hat{\tau_i}$ (δ_i and r_i are known after R_i is detected).

Note that while SB, DB, and UL models generate the estimate $\hat{h_{i+1}}$ right after client request h_i is detected, UL+DL model must wait for server response R_i to compute the features and perform the prediction. Moreover, while SB and DB models do not need to be trained, both UL and UL+DL require a training phase as well as the tuning of RF hyper-parameters, similarly to what described in Section III-A. To compare the models performance, we compute the Cumulative Distribution Functions (CDFs) of the normalised errors when either Blind or Audio or Video approach is adopted. Note that when Blind is selected, the prediction errors are normalised to the average inter-request time observed in $\mathcal{Y}$. Conversely, when Audio or Video is selected, normalisation is done with respect to requests of the same type, i.e., audio or video respectively. For the sake of clarity, we summarise in Table II the average inter-request times observed in such three cases, for either QUIC or TCP transport protocols. As one can see in Figures 6, 7, and 8, regardless of the approach, UL and UL+DL models perform better than the baselines, with UL+DL model outperforming all the others. In details, when uplink requests are carried by QUIC traffic, UL+DL improves the 80-th percentile of normalised error by 54 %, 88 %, and 51 % for Blind, Audio and Video approach respectively, whereas it is decreased of 29 %, 75 %, and 39 % when TCP is used. Let us now compare the CDFs of the normalised errors for Blind, Audio and Video approaches when UL+DL prediction model, i.e., the best performing one, is selected. As shown in Figure 9, pipe-lining request type classification and prediction of next request arrival time improves the final prediction performance, regardless of the transport protocol used. In fact, for both QUIC and TCP traffic flows, both

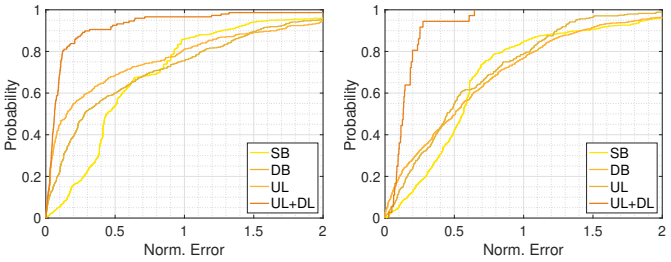

Fig. 7. Audio, QUIC (left) and TCP (right) traffic flows.

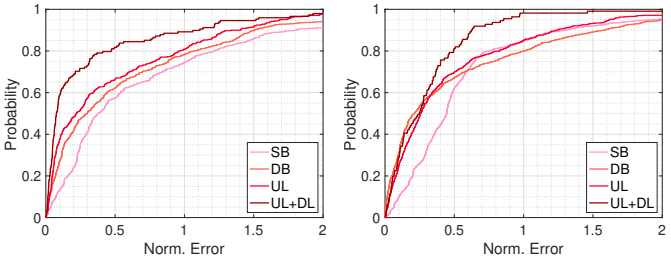

Fig. 8. Video, QUIC (left) and TCP (right) traffic flows.

the CDFs for Audio (yellow line) and Video (red line) lay above the one of Blind (blue line), where largest gaps are observed in the case of audio requests. This is summarised in Table III, where we compare the average 50-th and 80-th percentile of the normalised error obtained from UL+DL and baseline models. Note that the errors are averaged with respect to the two cases of QUIC and TCP traffic flows (Table III also reports the absolute values of the errors in seconds). As one can see, UL+DL model outperforms the baseline, yielding an average improvement of 78 % and 70 % for 50-th and 80-th percentile, respectively. Moreover, referring to UL+DL model, we observe that Audio and Video approaches decrease the normalised 50-th percentile error by 52 % and 42 % with respect to Blind approach. This leads to comparable median absolute errors between Audio, Video, and Blind, ranging between 0.68 s and 0.76 s. As far as the 80-th percentile of normalised error is concerned, Audio outperforms Blind by 72 %, whereas Video and Blind approaches yield comparable values. These results lead to the following conclusions. First, regardless of the approach, prediction performance improve when both uplink (i.e., CR) and downlink (i.e., SR) related features are included in the model. Second, a good strategy to predict the arrival of next uplink request is to first classify the last observed request and then perform prediction. This approach has a twofold advantage: i) it improves the prediction performance (i.e., the accuracy of the answer *when* next request will come) and ii) it provides with the knowledge of *what* type next request will be, i.e., whether for audio or video contents. This can be exploited by a streaming provider, which can infer the amount of data which will be delivered by the server and proactively adjust the network configuration, if needed. Finally, we observe that predicting the arrival of next request for video contents is more complex than doing the same for audio contents.

TABLE III
ABSOLUTE AND NORMALISED PREDICTION ERRORS OF SB VS UL+DL REGRESSION MODELS FOR BLIND, AUDIO, AND VIDEO APPROACHES.

Model	**SB**				**UL+DL**			
Percentile	**50-th**		**80-th**		**50-th**		**80-th**	
	Abs. (s)	**Norm. (%)**	**Abs. (s)**	**Norm. (%)**	**Abs. (s)**	**Norm. (%)**	**Abs. (s)**	**Norm. (%)**
Blind	2.25	43.06	6.17	118.27	0.68	13.16	2.33	45.27
Audio	5.54	50.45	11.63	105.84	0.69	**6.30**	1.33	**12.15**
Video	3.41	34.29	11.03	110.86	0.76	**7.65**	4.44	**44.63**

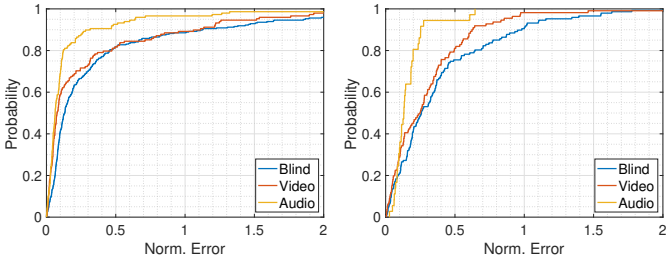

Fig. 9. Blind (blue) vs Audio (yellow) vs Video (red) approach, for QUIC (left) and TCP (right) traffic flows.

IV. CONCLUSIONS AND FUTURE WORKS

In this paper we present a machine-learning based monitoring architecture able to perform a twofold task, namely i) classification of HAS uplink request type and ii) prediction of next request arrival time. The features used to perform these tasks have several advantages: i) they are extracted from encrypted traffic with very low computational complexity, ii) they are extracted online, i.e., as the network traffic flows and iii) they can be easily updated for each new request issued by the client. We evaluate the performance of the system on a dataset comprising 900 YouTube video sessions and more than 15,000 HTTP request-response exchanges, extracting features for both directions of the traffic. Results show that our system is able to i) differentiate audio and video flows with excellent accuracy (greater than 95 %), and ii) leverage request type information to dramatically increase the performance of next request arrival prediction task. This information is crucial for network operators to infer the amount of data which will be delivered by the server and proactively adjust the network configuration when needed. Future works will regard the evaluation of the system performance when users interactions and advertisement clips playback are enabled during video sessions. As for real-time scenarios, main challenges regard the live packet-capturing process and the monitoring of multiple parallel traffic flows for single or multi-clients instances.

REFERENCES

[1] Sandvine, "The global internet phenomena report covid-19 spotlight," May 2021. [Online]. Available: https://www.sandvine.com/phenomena

[2] "Cisco annual internet report white paper," March 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] A. Okic and A. E. Redondi, "Optimal resource allocation in c-ran through dsp computational load forecasting," in *31st Annual Int. Symp. on Personal, Indoor and Mobile Radio Comm.*, 2020.

[4] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5g network slicing resource utilization," in *IEEE Conf. on Computer Communications*, 2017.

[5] M. Joshi and T. H. Hadi, "A review of network traffic analysis and prediction techniques," *arXiv preprint arXiv:1507.05722*, 2015.

[6] J. Barros, M. Araujo, and R. J. Rossetti, "Short-term real-time traffic prediction methods: A survey," in *Intl. Conf. on Models and Technologies for Intelligent Transportation Systems*, 2015.

[7] D. Tsilimantos, T. Karagkioules, and S. Valentin, "Classifying flows and buffer state for youtube's http adaptive streaming service in mobile networks," in *9th ACM Multimedia Systems Conf.*, 2018.

[8] F. Loh, F. Wamser, C. Moldovan, B. Zeidler, D. Tsilimantos, S. Valentin, and T. Hoßfeld, "Is the uplink enough? estimating video stalls from encrypted network traffic," in *IEEE/IFIP Network Operations and Management Symp.*, 2020.

[9] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura, "emimic: Estimating http-based video qoe metrics from encrypted network traffic," in *Network Traffic Measurement and Analysis Conf.*, 2018.

[10] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Stream-based machine learning for real-time qoe analysis of encrypted video streaming traffic," in *22nd Conf. on innovation in clouds, internet and networks and workshops (ICIN)*, 2019.

[11] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "Vicrypt to the rescue: Real-time, machine-learning-driven video-qoe monitoring for encrypted streaming traffic," *IEEE Trans. on Network and Service Management*, 2020.

[12] F. Loh, F. Poignée, F. Wamser, F. Leidinger, and T. Hoßfeld, "Uplink vs. downlink: Machine learning-based quality prediction for http adaptive video streaming," *Sensors*, 2021.

[13] I. Orsolic and M. Seufert, "On Machine Learning based Video QoE Estimation Across Different Networks," in *16th Intl. Conf. on Telecommunications*, Zagreb, Croatia, 2021.

[14] I. Orsolic and L. Skorin-Kapov, "A framework for in-network qoe monitoring of encrypted video streaming," *IEEE Access*, 2020.

[15] C. Gutterman, K. Guo, S. Arora, T. Gilliland, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time qoe metric detection for encrypted youtube traffic," *ACM Trans. on Multimedia Computing, Communications, and Applications (TOMM)*, 2020.

[16] J. Lee, S. Lee, J. Lee, S. D. Sathyanarayana, H. Lim, J. Lee, X. Zhu, S. Ramakrishnan, D. Grunwald, K. Lee *et al.*, "Perceive: Deep learning-based cellular uplink prediction using real-time scheduling patterns," in *18th Intl. Conf. on Mobile Systems, Applications, and Services*, 2020.

[17] A. Schwind, M. Seufert, Ö. Alay, P. Casas, P. Tran-Gia, and F. Wamser, "Concept and Implementation of Video QoE Measurements in a Mobile Broadband Testbed," in *IEEE/IFIP Workshop on Mobile Network Measurement (MNM)*, Dublin, Ireland, 2017.

[18] F. Wamser, M. Seufert, P. Casas, R. Irmer, P. Tran-Gia, and R. Schatz, "YoMoApp: a Tool for Analyzing QoE of YouTube HTTP Adaptive Streaming in Mobile Networks," in *European Conf. on Networks and Communications (EuCNC)*, Paris, France, 2015.

[19] M. Seufert, "Quality of Experience and Access Network Traffic Management of HTTP Adaptive Video Streaming," Doctoral Thesis, University of Würzburg, 2017. [Online]. Available: https://opus.bibliothek.uni-wuerzburg.de/files/15413/Seufert_Michael_Thomas_HTTP.pdf

[20] S. C. Madanapalli, H. Habibi Gharakhieli, and V. Sivaraman, "Inferring netflix user experience from broadband network measurement," in *2019 Network Traffic Measurement and Analysis Conf. (TMA)*, 2019.