Exploring the Usefulness of Machine Learning in the Context of WebRTC Performance Estimation

Doreid Ammar*, Katrien De Moor†, Lea Skorin-Kapov‡, Markus Fiedler§, Poul E. Heegaard†
*em**lyon business school**, Ecully, France
ammar@em-lyon.com
†NTNU - Norwegian University of Science and Technology, Trondheim, Norway
{katrien.demoor|poul.heegaard}@ntnu.no
‡University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
lea.skorin-kapov@fer.hr
§Blekinge Institute of Technology, Dept. of Technology and Aesthetics, Karlshamn, Sweden
markus.fiedler@bth.se

***Abstract*—We address the challenge faced by service providers in monitoring Quality of Experience (QoE) related metrics for WebRTC-based audiovisual communication services. By extracting features from various application-layer performance statistics, we explore the potential of using machine learning (ML) models to estimate perceivable quality impairments and to identify root causes. We argue that such performance-related data can be valuable and informative from a QoE assessment point of view, by allowing to identify the party/parties in a call that is/are experiencing quality impairments, and to trace the origins and causes of the problem. The paper includes case studies of multi-party videoconferencing that are established in a laboratory environment and exposed to various network disturbances and CPU limitations. Our results show that perceivable quality impairments in terms of video blockiness and audio distortions may be estimated with a high level of accuracy, thus proving the potential of exploiting ML models for automated QoE-driven monitoring and estimation of WebRTC performance.**

***Index Terms*—Quality of Experience (QoE), WebRTC, machine learning, video-blockiness, audio distortion**

I. Introduction

Despite the availability of numerous applications that offer audiovisual communication services, strict low-latency and high volume requirements continue to impose challenges in meeting end user Quality of Experience demands. While commercial proprietary video conferencing solutions have a limited reach, solutions based on the Web Real-Time Communications (WebRTC) standards and APIs (https://webrtc.org) are increasingly gaining momentum and enabling browser-to-browser real-time communication with built-in real-time audio and video functions [1]. This is reflected in the adoption of numerous free of charge applications, such as Google Hangouts, Facebook Messenger, appear.in, etc., which can be used in different contexts (*e.g.,* business vs. leisure).

From a Quality of Experience (QoE) point of view, end users may experience different types of technical problems, going hand in hand with different types of impairments (*e.g.*, video freezes, bad audio, no audio) during a conversation, which may hinder the degree and nature of interaction between different parties involved. Such circumstances may also trigger affective responses (*e.g.*, user frustration, stress, etc.), cognitive complexities and communication problems (*e.g.*, misunderstandings between parties in the call) and behavioural reactions (either instantaneously, *e.g.*, by switching off video, reconnecting, etc., or over time, *e.g.*, by not re-using the application after a number of problematic sessions). For application and service providers, it is of crucial importance to monitor service quality and identify root causes of perceived impairments, so as to provide the basis for potential problem mitigation and QoE control (e.g., dynamic service adaptation in light of the number of parties, devices used, network connections, etc.).

In this paper, we focus on exploring the usefulness of machine learning (ML) in the context of WebRTC performance estimation and root cause analysis from a service provider perspective. Previous work [2], [3] has investigated the usefulness and limitations of the WebRTC performance statistics gathered by Google Chrome for QoE studies. We now go one step further and aim to predict the occurrence of perceivable quality impairments for the user and to identify root causes underlying these impairments by using WebRTC performance statistics as input features for ML-based classifiers. We here limit the scope to two particular types of perceivable impairments, namely video blockiness and audio distortion, and to well-interpretable results and findings gathered through the use of a research version of the WebRTC application appear.in.

The paper is organized as follows: In Section II, we give a brief overview of relevant previous work addressing QoE monitoring for WebRTC, and in particular the use of ML-based models. Section III describes the data collection and feature extraction, followed by a case study in Section IV. Finally, Section V concludes the paper and shares a number of lessons learned and directions for future work.

II. Background and related work

To date, a number of papers have addressed the issue of how to monitor WebRTC performance, both from a QoS and QoE perspective. In practice, and often complementary

to the collection of user feedback, providers rely on objective performance monitoring tools to estimate perceived quality degradations, using QoE models as a basis. Data may be collected at different levels, including network-level QoS measurements (*e.g.*, packet loss, delay, jitter), as well as application-level measurements (*e.g.*, audio/video bitrate, resolution, and framerate). Quality estimations may rely on analytical QoE models (*e.g.*, derived using regression analysis), or on other ML-based approaches to tackle underlying relationships.

Garcia et al. [4] present a general methodology and tool for black-box testing of WebRTC applications and services. A WebRTC application is run in a containerized cloud environment, where various end-to-end tests are run, and QoS and QoE indicators are captured. While such a testing framework could be used to further explore the QoS-to-QoE mapping, this is not explicitly addressed by the authors. In their subsequent work, Garcia et al. [5] discuss a set of measurable factors to estimate the QoE of a WebRTC application, namely call establishment time, end-to-end delay, audio quality, video quality, and audiovisual quality.

Focusing on the impact of network performance, Jansen et al. [6] investigate the effects of latency, packet loss, and bandwidth on the performance of WebRTC-based videoconferencing in both synthetic and real wired and wireless environments. In particular, they focus on the impact of the Google Congestion Control (GCC) algorithm, utilized by WebRTC to provide congestion control for real-time communications over UDP. Addressing multi-party WebRTC calls in a mobile context, Vucic and Skorin-Kapov [7] further study the impact of packet loss and GCC on perceived audio quality, video quality, and overall QoE.

In a generic approach, Spetebroot et al. [8] aim to predict expected voice and/or video quality of various target applications (e.g., Skype) based on the analysis of performance at the device and network level, without actually requiring the target application to be running. For such purposes, different ML models relating measurable performance to QoE metrics are needed for different target applications.

The question arises as to what measurable data may be utilized (and how) for the purpose of detecting QoE-related metrics for WebRTC-based audiovisual communication services. To that extent, De Moor et al. [9] report on a subjective study aimed to explore the usefulness of different types of data (self-report data, peripheral physiological data, and application-level performance statistics) in providing indications of users QoE and affective state in the context of WebRTC. Sulema et al. [10] set up WebRTC video streams in a mobile broadband network, and utilize the open access MONROE platform providing access to hundreds of nodes connected to different operators across Europe. The authors analyzed collected QoS measurements, and further used a combination of subjective tests and ML-based models to assess overall QoE. However, details on the ML-based models are missing.

Further looking to relate measurable QoS metrics to QoE, Yan et al. [11] conduct measurements in a WiFi network and build an ML model to infer whether QoE is acceptable or not in the next time window based on current QoS metrics (including Round-Trip Time, Link Quality, and Received Signal Strength Indicator (RSSI)). As a proxy measure for QoE, they rely on detecting video freezing events.

While previous papers have studied various approaches in monitoring WebRTC QoE-related metrics, in this paper we provide novel insights in terms of utilizing WebRTC application performance statistics exposed by the WebRTC API to estimate video blockiness, audio distortions, and to identify root causes of problems. A number of tests are performed under various emulated network conditions, and results show the usefulness of utilizing ML-based models in WebRTC performance estimation. We cast the problem as a supervised discrete classification problem, as described in further detail in the following section. Decision trees are shown to provide an intuitive understanding of underlying relationships.

III. Methodology

A. Overview

As introduced earlier, with this paper we aim to explore whether and how machine learning may be useful for gaining insight into potential QoE-related issues in the context of WebRTC. More specifically, we collected different types of data related to the use of a research version of the WebRTC-based application called appear.in.

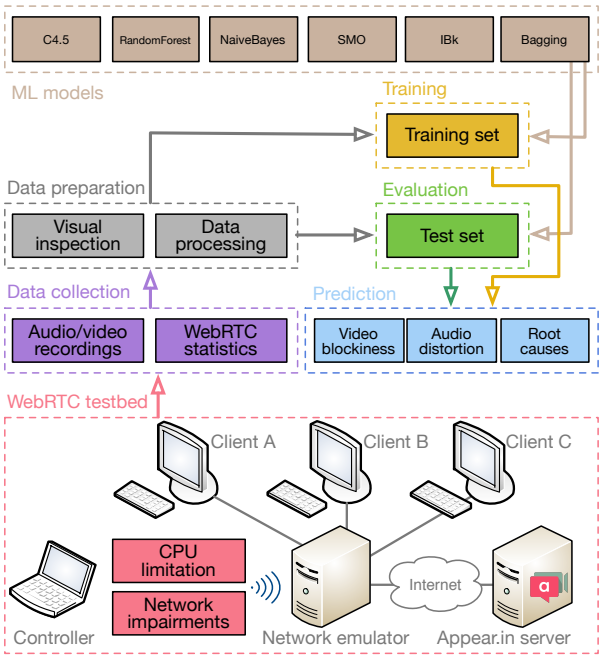


Fig. 1: Overview of laboratory setup and test procedure

Figure 1 provides a high-level overview of the general set-up and procedure that are applied in this paper. The gathered data consisted of application-level WebRTC performance statistics and audio/video recordings collected during multiple two- and three-party audiovisual calls. Data from two- and three-party video calls was collected under different CPU limitations and

network impairments, using the WebRTC testbed described in [12]. Previous work [2], [3] has provided insights into the usefulness (and limitations) of WebRTC performance statistics gathered by Google Chrome and in the root causes of technical impairments that are perceivable (*e.g.*, visually and/or auditory) by users. The work presented in this paper goes a step further and applies ML algorithms to the gathered datasets in order to estimate video blockiness, audio distortions, and to identify root causes of problems. Before turning to the more detailed description of the different steps as indicated in Figure 1, namely data collection, data preparation, model training and evaluation of the obtained model, we briefly discuss the WebRTC testbed used to run different test scenarios and collect the corresponding data.

The WebRTC testbed used in this work consists of a research version of the WebRTC-based application called *appear.in* [13], a *network emulator*, and a *controller*. The testbed provides video conferencing service by *appear.in* in a controlled environment. The lower part of Figure 1 illustrates a simplified version of the testbed with a 3-party setup. It is important to note that calls are established in a peer-to-peer manner. The *network emulator* relies on NetEm to control network conditions, introducing (i) bandwidth throttling; (ii) delay and delay variation (jitter); (iii) packet loss and packet burst losses, hereafter referred to as mean lost burst size (mlbs).

B. Data collection

In this sub-section, we briefly explain which type of data are collected and elaborate on the used test profiles and generated datasets.

1) WebRTC performance statistics: In WebRTC, media specific (video, audio, screen sharing) streams are transmitted over a *peer-connection* using Real-time Transport Protocol (RTP) over UDP. On the other hand, non-media data types are transmitted using Stream Control Transmission Protocol (SCTP) encapsulated in Datagram Transport Layer Security (DTLS) over UDP. A connection is assigned a unique identity, SSRC ID, defined as a *track*. A *track* is a media specific *data channel* between two peers (browsers).

Figure 2 shows a three-party WebRTC video conference call. Between each pair of peers, four tracks are exchanged (two video and two audio), each of which is identified by a unique SSRC ID. The SSRC ID links the three parties in a 3 Peer Connections, including 6 audio tracks (3 for sending and 3 for receiving) and 6 video tracks (3 for sending and 3 for receiving)[1].

The World Wide Web Consortium (W3C) has specified W3C WebRTC statistics [14], by defining objects that allow access to the statistical information about a real-time peer connection (RTCPeerConnection). The statistics can be classified with respect to codec used, certificate used, transport protocol used, and contains counters, such as number of peer connections.

[1]In *appear.in* screen sharing is possible and activating this will create 6 additional video tracks

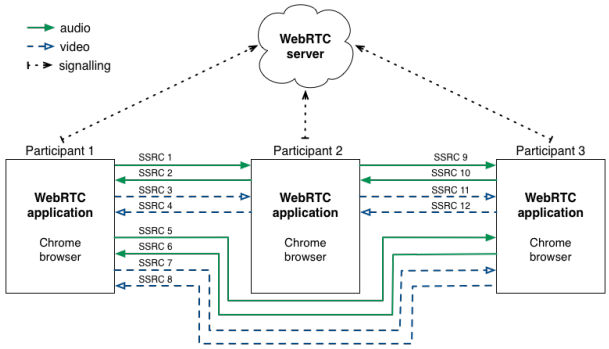


Fig. 2: A three-party video conference call opens twelve tracks

In the testbed described in [12], we collect WebRTC statistics using Google Chrome *webrtc-internals* (chrome://webrtc-internals), which enables observation of the performance of the WebRTC connections locally in the browser. The statistics are collected per browser. Despite all the challenges, previous work has demonstrated the potential of using these statistics to study QoE aspects of WebRTC services [2], [3].

Other performance measurement tools (such as Wireshark and even probing nodes) are complementary to WebRTC API statistics and may be used to trace network aspects like packet delay, jitter, and bandwidth, while network performance measurements can be used for QoE troubleshooting and network diagnosis.

2) Audio and video recordings and coding of the material: The transmitted audio and video contents of each individual participant are recorded on both sending and receiving ends using SimpleScreenRecorder (on Linux). This allows us to replay the session and analyse the recordings. The audiovisual recordings contained the following impairments: video blockiness, video freeze, jerkiness, flickering, delay in audio-video synchronisation, no audio, audio distortion. In this paper, we focus as mentioned on video blockiness and audio distortion, and the audiovisual recordings for each conversation were inspected and annotated from the perspective of each party in the different calls. In the context of this study, 55 audio and video recordings were inspected and annotated.

We define blockiness as the appearance of a block structure in a video, either due to lossy compression and block-based coding schemes [15], or transmission errors, such as the loss of (or loss within) a reference video frame [16]. As the thresholds between the different levels are fuzzy and cannot be precisely determined in manual annotations, we applied a two level categorisation of the blockiness: *acceptable* (i.e., no or very low blockiness) and *not acceptable* (i.e., high blockiness). No appropriate tool for automated annotation was found, so the annotation was done manually by two independent coders. They identified perceivable impairments (with a time granularity of 1 second) and annotated them. For future and upscaled studies however, automated annotation solutions need to be developed. After having coded the material independently, the annotations from both coders were manually compared and integrated. For a number of cases where the annotations

TABLE I: Experiment scenarios and selected values for network impairments and CPU limitations.

Scenario	Parameter	Values	Profile
S_1	–	–	G
S_2, S_3	plr $(mlbs)$	10% (1.5 or 3)	V
S_4, S_5	plr $(mlbs)$	20% (1.5 or 3)	V
S_6, S_7	$delay$ $(jitter)$	500 (300 or 500) ms	AV
S_8, S_9, S_{10}	cpu	55%, 60% or 70%	A

diverged, the recording was checked again by both coders and discussed in order to reach an agreement.

3) Test profiles and generated datasets: The datasets were collected from twenty-five WebRTC-based multi-party video conversations, including twenty 2-party calls and five 3-party calls, each with a call duration between 4 and 5 minutes. The conversations took place in a controlled lab environment and under different technical conditions. The conversation partners were located in separate rooms with not identical, but very similar dimensions, lighting and background conditions. In total, three rooms were used and equipped with identical desktop computers.

More specifically, by means of the WebRTC testbed, we consider a number of different call scenarios. These are (i) *no impairments*, (ii) *realistic technical impairments*, e.g., packet losses, packet delays, and CPU limitations.

Table I provides an overview of the experiment scenarios and the selected values for the introduced network impairments, as well as the CPU limitations. Each of these scenarios leads to different types and gradations of perceivable quality impairments at the user side (*e.g.*, video blockiness, video freezes, bad audio). Hereafter, we refer to the produced audio and video qualities as "profile". For the work presented in this paper, we used the following four types of profiles:

G: *Good conditions* (S_1) - no perceivable distortion,
V: *Video distortion* (S_2 - S_5) - video distortion only,
A: *Audio distortion* (S_8 - S_{10}) - audio distortion only,
AV: *Audio and video distortion* (S_6, S_7) - both audio and video distortion.

It is worth mentioning that we ran a series of real-life calls (around 50 calls with different call duration, ranging from 4 minutes to 45 minutes) in which we focused on two- and three-party video conferencing using the appear.in application. Parties involved in these calls experienced various network conditions. The collected WebRTC performance statistics are used as a reference and further evaluated in order to gain a better understanding of real-life network impairments. Throughout the calls where parties have bad network conditions (*e.g.*, week spots of WLAN connectivity), we observed severe packet losses, ranging from 0% to 35% (with few spikes going above 50% over time intervals of length 10 seconds), and a high-delay spiking up to nearly 1250 ms. These observations were used as a reference for selecting values for the introduced network impairments in our experimental scenarios. Therefore, we reproduced calls in which parties would experience good or poor network conditions (i.e., with many critical phases of heavy audiovisual quality degradation).

C. Preparing the data for analysis using machine learning

Before the training phase and application of machine learning algorithms could start (as visualised in Figure 1), the data had to be prepared and pre-processed. This phase consisted of two main tasks, (i) pre-processing of blockiness and audio distortion annotations, and (ii) pre-processing of WebRTC statistics.

The collected data was prepared as input for the WEKA tool [17], a java library containing implementations of all ML algorithms tested in this paper. The WebRTC statistics and the coded material were transformed to a format fitting the WEKA tool (*i.e.*, one file per peer connection and per impairment type, classified into the respective categories). This file contains, the WebRTC statistics that were described in Section III-B1, and the coded material (*i.e.*, video blockiness, audio distortion and root causes). It is worth noting that, the WebRTC statistics (*e.g.*, *Bucket delay*, *Bitrate*, *Encoding*, *Frame rate*, the *Frame Scale* (scale of the video *Frame size*, where *Frame size* is equal to *Frame width* $\times$ *Frame height*), *Packet loss ratio*, *Nack messages*, etc.) are collected/computed over a time scale of 1 second. Furthermore, the coded materials are classified (with a time scale of 1 second) into the respective categories: (i) acceptable level of blockiness and unacceptable level of blockiness when we deal with video blockiness; (ii) no audio distortion and audio distortion when we deal with audio distortion; (iii) and good conditions, network impairments and CPU limitations for root causes classifications.

D. Training a model

For the training step, a training set (See Figure 3) from seventeen conversations was used. More specifically, the log files were recorded from fourteen 2-party calls and three 3-party call. These conversations were conducted using scenarios S_1 to S_{10}. The training set is used to build the decision trees for video blockiness, root causes and audio distortion shown in Figures 6, 7 and 8.

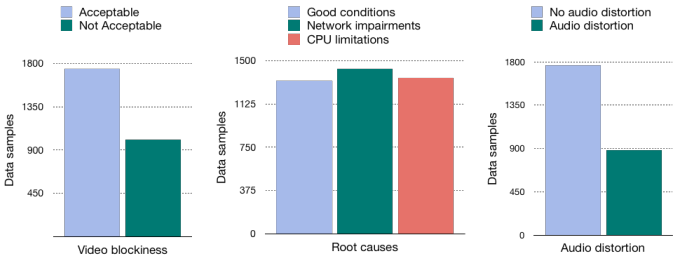


Fig. 3: Number of data samples used in training the ML models.

On the other hand, we have selected six widely-used ML algorithms, namely ***C4.5*** decision tree learner, ***RandomForest*** that operates by constructing a multitude of decision trees, standard probabilistic ***NaiveBayes*** classifier, sequential minimal optimization algorithm for support vector regression

(***SMO***), k-nearest-neighbors classifier (***IBk***) and ***Bagging***. For more details, please refer to [17]. For each of the six tested algorithms, models were built by using a tenfold cross-validation scheme. This scheme splits randomly the dataset into 10 subsets in which the class is represented in approximately the same proportions as in the full dataset. Then it uses nine of the subsets to train the model and the last one to cross-validate the model on. This learning procedure is repeated a total of 10 times for all the combinations of the subset. Finally, the results of the cross-validation scheme are averaged to yield an overall estimation of model's performance.

E. Feature Selection

In our work, we use various feature selection methods to extract the most useful features to train on among existing *webrtc-internals* features. The selected features are:

Bucket delay: "bweforvideo-googBucketDelay" – The bucket delay is defined as "... the time since the oldest queued packet was enqueued" [18].

Encoding: "send-googEncodeUsagePercent" – The average encode time (in millisecond) divided by the average time difference between incoming captured frames [18].

Frame scale: "send-googFrameHeightSent" and "send-googFrameWidthSent" – The height and the width of the video frame in pixels.

Frame rate: "send-googFrameRateSent" – The rate at which consecutive frames are displayed in an animated display.

Nacks: "send-googNacksReceived" – The cumulative number of received negative-acknowledgements (nack) messages. Nack packets are received by the sender and sent by the receiver using RTCP, as defined further in [19].

Packets lost: "send-packetsLost" – The cumulative number of packets lost.

Input level: "send-audioInputLevel" – The audio input level.

Output level: "recv-audioOutputLevel" – The audio output level.

Jitter: "send-googJitterReceived" – The jitter.

Jitter buffer: "recv-googJitterBufferMs" – The jitter buffer.

Bitrate: "Conn_audio_bitsReceivedPerSecond" – The received audio bitrate.

ERLE: "send-googEchoCancellationReturnLossEnhancement" – The echo Cancellation Return Loss Enhancement.

EC delay: "send-googEchoCancellationEchoDelayMedian" – The echo Cancellation delay.

F. Evaluating the model

To strengthen the performance analysis (*e.g.*, test the stability, prevent overfitting, generalize the results, etc.), we use the models trained from the training set and test them using a test set (see Figure 4) that contains logs (never been used in training) from five 2-party calls two 3-party calls using the following scenarios: S_1, S_3, S_5, S_7, S_8, S_9 and S_{10}. The later scenarios correspond to all four different profiles.

The following test set is used for the performance analysis of the decision trees for video blockiness, root causes and audio distortion shown in Figures 6, 7 and 8.

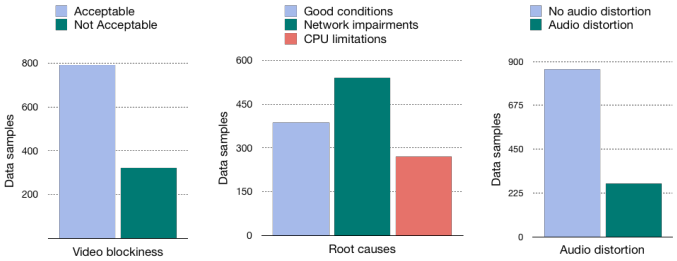


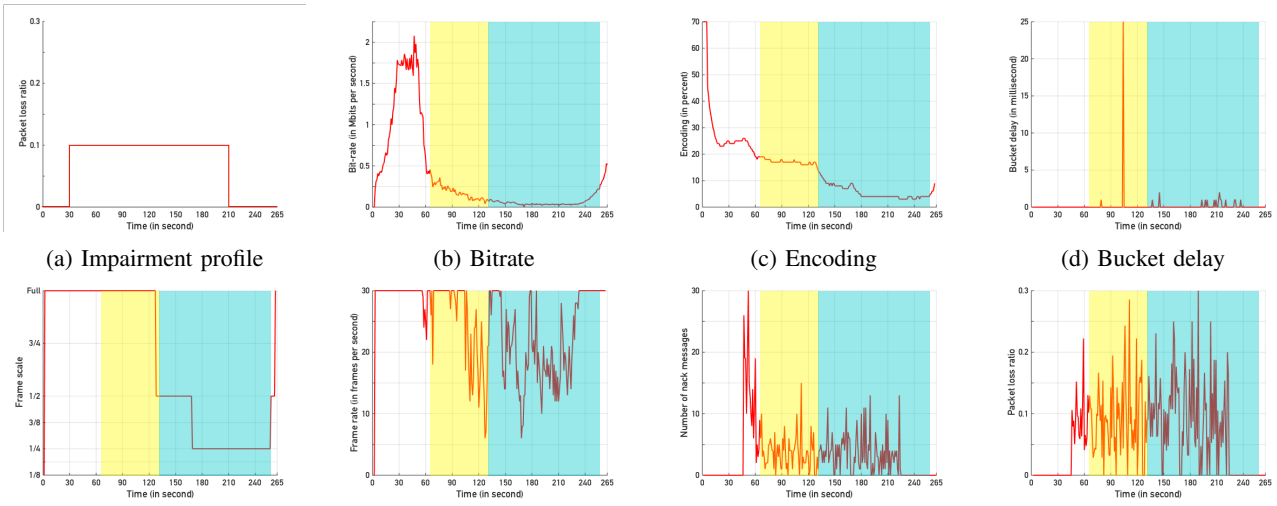
Fig. 4: Number of data samples used in testing the ML models.

IV. Case Study

In this section we discuss the ML modeling results, implications and interpretations. Actually, the WebRTC statistics capture the joint impact of network impairments and the Google Congestion Control (GCC) algorithm included in the Google Chrome browser and implemented in the WebRTC project [20], [21]. The GCC algorithm is specifically designed to target real-time streams such as telephony and video conferencing [22], [23], [24], thereby trying to fully utilize the bottleneck link while keeping queuing delay small [24]. According to [20], [21], the GCC algorithm implements both a delay-based and a loss-based controller, which is run on the sender side in response to feedback from the receiver. Considering the specific case of packet loss that we use in our case studies, we note that when a threshold of 10% packet loss is detected by the sender, GCC performs a new bandwidth estimate and subsequently invokes stream adaptation, including bitrate, resolution, and finally framerate adaptation. The interested reader is referred to [20], [21] and [24] for further details regarding GCC. In the sequel, we will see the impacts of network impairments, GCC algorithm and user perceivable quality metrics (such as video blockiness and audio distortion) in the ML models. Although various types of ML classifiers were trained and found to provide comparable results (e.g., Random Forest, NaiveBayes, SMO, k-nearest-neighbors, Bagging), these results are not included due to space limitations. We include only results obtained with the C4.5 decision tree[2] as they are easily interpreted [3] It is also worth noting that, the data was randomly split into training and test data. More specifically, around 70% (i.e., 18 conversations) of the data were used for training the ML model and the remaining data (7 conversations) was used for testing the performance of the ML model. In this work, we limited the size of the dataset to 25 conversations (55 recordings – nearly 4 hours of recordings) since obtaining additional data samples (conversations) and annotating the perceivable quality impairments is a time-consuming task.

[2]The following tuning is used for the C4.5 decision tree: batch size (100); confidence threshold for pruning (0.25); minimum number of instances permissible at a leaf (25); size of the pruning set (3); pruning is performed (True); subtree raising operation when pruning (True); use of Laplace (False); and use of MDL correction (True).

[3]Training and test data: https://github.com/doreidammar/webrtc-statistics.git



(a) Impairment profile (b) Bitrate (c) Encoding (d) Bucket delay

(e) Frame scale (f) Frame rate (g) Nack messages (h) Packet loss ratio

Fig. 5: A selection of collected metrics on one client over time, corresponding to profile V and scenario S_2. A packet loss ratio of 10% is introduced after 30s, lasts for 3 min, followed by 30s of no impairments. Phases with *no*, *low*, and *high* degree of video blockiness are highlighted in white, yellow, and blue, respectively. The white and yellow areas combined correspond to what we refer to as *acceptable* blockiness, while the blue areas correspond to *unacceptable* blockiness.

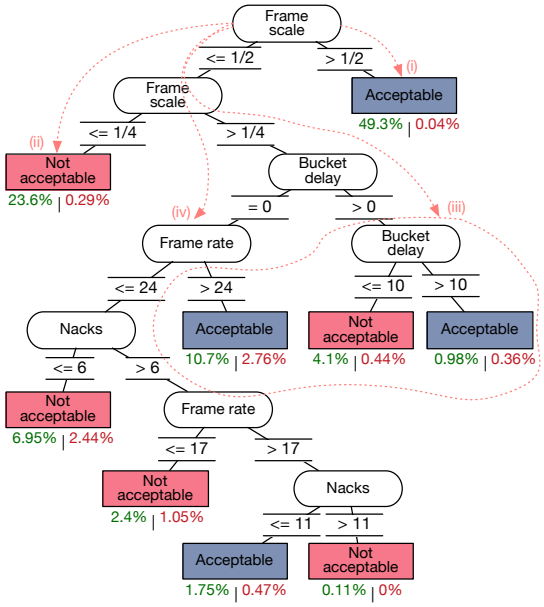


Fig. 6: Decision tree for acceptability in terms of blockiness. Green percentages indicate the overall share of correct classifications ("trues"), red percentages the overall share of incorrect classifications ("falses") through the corresponding rule.

A. Results and Observations

Figure 6 shows the decision tree for the *acceptability in terms of blockiness* obtained through the training phase, with the following confusion matrix:

actual class \ *predicted class*	Acceptable	Not acceptable
Acceptable	59.85%	3.49%
Not acceptable	4.40%	32.25%

The high accuracy of the classification is seen from the diagonal dominance ($59.85 + 32.25 = 92.10\%$), with misclassifications less than 5 % of each kind.

The major decision criterion, at the root of the tree in Figure 6, is the frame scale subject to adaptation by the GCC algorithm. We observe: (i) Frame scales of more than 1/2 (240p) are indicated as acceptable in terms of blockiness, which applies to almost half of all cases, with an almost negligible misclassification ratio. Taking the actions of the GCC into account, large frame scales imply good network conditions. (ii) Frame scales of less than 1/4 (120p) are deemed to be not acceptable in terms of blockiness, which applies to almost 24 % of the cases, again with a small misclassification ratio. (iii) For frame scales in-between the above cases (i.e. videos of 180p and 240p), the bucket delay plays a key role, but even other parameters increase in relevance. In case of zero bucket delay, a frame rate of more than 24 fps indicates an (in terms of blockiness) acceptable quality (applies to 10.7 % and misclassifies in 2.76 % of the cases). (iv) The majority of the remaining cases are deemed not to be acceptable in terms of blockiness. These observations are in line with Figure 5, in particular with subfigure 5e. The factor two between the stimuli-related frame scale threshold values 1/4 and 1/2 aligns with the Weber-Fechner Law [25].

We now turn our attention to the decision tree for *root causes for blockiness* and origins of performance issues, see Figure 7, with diagonal-dominant confusion matrix:

actual class \ *predicted class*	Good cond.	Network imp.	CPU limit.
Good conditions	30.82%	0.41%	1.12%
Network impairments	2.60%	32.18%	0.00%
CPU limitations	0.32%	0.02%	32.52%

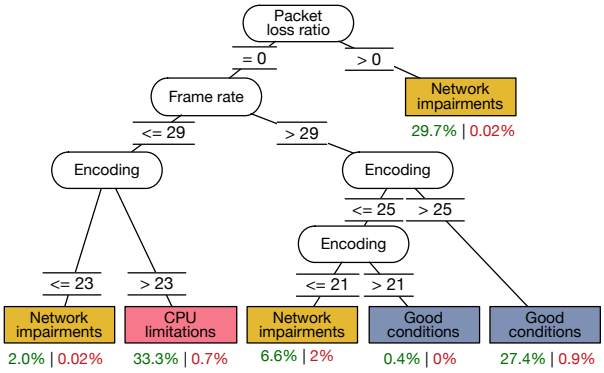


Fig. 7: Decision tree for root causes. Green percentages indicate the overall share of correct classifications ("trues"), red percentages the overall share of incorrect classifications ("falses") through the corresponding rule.

We observe the following: (i) *Good conditions* are mainly indicated for vanishing packet loss, the nominal frame rate, and good encoding performance. (ii) *CPU limitations* are indicated for vanishing packet loss, a sub-nominal frame rate together with a high encoding value, pointing at sub-optimal yet good-enough delivery on network level. (iii) *Network impairments* are mainly observed in combination with non-vanishing packet loss, and to a small extent for low frame rates in combination with a low encoding value, all pointing at delivery issues.

Finally, we take a look at the decision tree for *audio distortions*, shown in Figure 8, with confusion matrix:

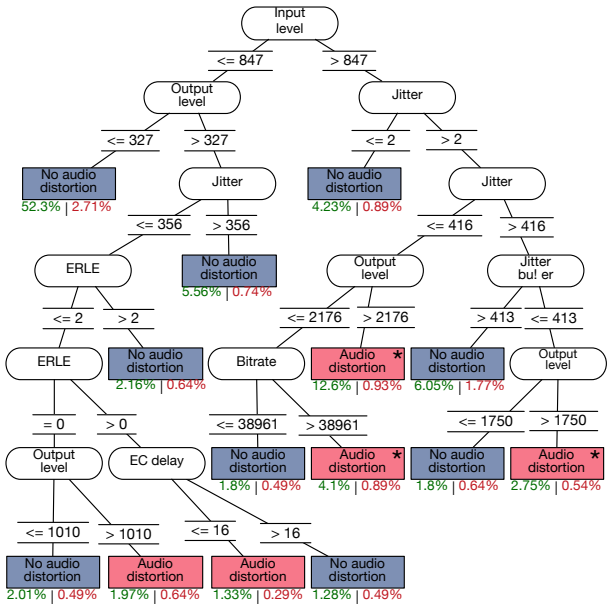


Fig. 8: Decision tree for audio distortion. Green percentages indicate the overall share of correct classifications ("trues"), red percentages the overall share of incorrect classifications ("falses") through the corresponding rule. The dominating distortions are marked with an asterisk ⋆.

TABLE II: Comparison of accuracy, precision, recall and F-score provided by the C4.5 algorithm, for acceptability in terms of blockiness and related, root causes, as well as for audio distortions.

2-level of blockiness – 2 classes: Acceptable | Not acceptable		
Training set	*Accuracy*	0.921
	Precision	0.932 | 0.902
	Recall	0.945 | 0.880
	F-score	0.938 | 0.891
Test set	*Accuracy*	0.976
	Precision	0.991 | 0.940
	Recall	0.975 | 0.978
	F-score	0.983 | 0.959
Root causes – 3 classes: Good conditions | Network impairments | CPU limitations		
Training set	*Accuracy*	0.955
	Precision	0.913 | 0.987 | 0.967
	Recall	0.953 | 0.925 | 0.990
	F-score	0.933 | 0.955 | 0.978
Test set	*Accuracy*	0.961
	Precision	0.955 | 0.989 | 0.921
	Recall	0.938 | 0.963 | 0.993
	F-score	0.946 | 0.976 | 0.962
Audio distortions – 2 classes: No audio distortion | Audio distortion		
Training set	*Accuracy*	0.829
	Precision	0.822 | 0.856
	Recall	0.951 | 0.586
	F-score	0.881 | 0.696
Test set	*Accuracy*	0.828
	Precision	0.864 | 0.680
	Recall	0.918 | 0.545
	F-score	0.990 | 0.605

predicted class

actual class	No audio distortion	Audio distortion
No audio distortion	63.40%	3.30%
Audio distortion	13.77%	19.53%

Indeed, the three top-most prominent conditions for audio distortions (marked with an asterisk ⋆ in the figure) come along with high input level, limited jitter, and either high output level or high received bit rate – or with jitter and jitter buffer mismatch in combination with a high output level. Obviously, high intensities bring about larger risks and sensitivities for audible distortions.

B. Classification Performance and Discussion

Table II shows the classification performance. With P as be the number of true positives, $\bar{P}$ the number of false positives, N the number of true negatives, and $\bar{N}$ the number of false negatives, respectively, we obtain the *accuracy* $\alpha = (P + N)/(P + \bar{P} + N + \bar{N})$, the *precision* $\pi = P/(P + \bar{P})$, the *recall* $\rho = P/(P + \bar{N})$ and the *F-score* $\phi = 2\pi\rho/(\pi + \rho)$. We observe that acceptability of blockiness and in particular the root causes yield high prediction performance values. In case of the audio distortions, the audio distortion prediction could be improved.

Indeed, the WebRTC statistics and the related decision trees reflect the behaviour of the service close to the user. The decision trees capture the combination of the control actions and of the underlying network and CPU problems (as far as they are handed through the stack, such as losses) that are of relevance to the user. Some technical problems are transformed into control actions on much larger time scales, that are well-perceivable by the users and nicely reflected in the decision trees, such as the role of the frame scale, cf. figures 6 and 5e.

V. Conclusions

In this paper, we explored the usefulness of machine learning to predict perceivable quality impairments and to identify the corresponding root causes in the context of a WebRTC-based, multi-party videoconferencing service. We focused specifically on the detection and estimation of audio distortion, video blockiness, and related root causes of performance issues by applying machine learning algorithms to a range of features extracted from WebRTC performance statistics. Despite the limitations of the current dataset, the results are promising and show that in most cases, a high level of accuracy can be achieved. With a large amount of statistics being collected during WebRTC calls, in particular in the case of multi-party calls, our approach enhances existing measurements with data analytics to provide deeper insight into user perceived service quality, which goes beyond the limited insights into QoE issues provided by network probes.

Take-aways from our work are as follows: (1) We demonstrated the potential of applying ML for identifying WebRTC performance issues and their origins. (2) We provide service providers/developers, having access to the aforementioned WebRTC API via the client browser and opportunities to adapt service delivery and coding parameters, with a tool to monitor and improve QoE. (3) Those service providers/developers will be able to track customer satisfaction, identify root causes of problems, and benchmark their networks against competitors.

In our ongoing and future work, the proposed approach will be extended towards a more comprehensive view by (i) running additional subjective studies (*e.g.*, considering a larger number of parties, asymmetric call conditions); (ii) including additional features (*e.g.*, extracted from physiological signals from the user, or data from network probes); (iii) using further ML algorithms and related hyper parameters; (iv) automated annotation approaches; and (v) considering a more complete spectrum of realistic and perceivable audio and video impairments, as well as their simultaneous impact, which calls for an extension of the current models and the development of multi-dimensional models.

Acknowledgments

The authors gratefully thank the appear.in team for their input and support. L. Skorin-Kapov's work has been supported in part by the Croatian Science Foundation under the project UIP-2014-09-5605 (Q-MANIC), and M. Fiedler's work by the Swedish Knowledge Foundation under the project "Big-Data@BTH" (grant: 2014-0032).

References

[1] H. Alvestrand, "Overview: Real time protocols for browser-based applications, draft-ietf-rtcweb-overview-19," IETF, Tech. Rep., 2017.

[2] D. Ammar, K. De Moor, M. Xie, M. Fiedler, and P. E. Heegaard, "Video QoE killer and performance statistics in WebRTC-based video communication," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE) (IEEE ICCE 2016)*, Ha Long Bay, Vietnam, Jul. 2016.

[3] D. Ammar, P. Heegaard, M. Xie, K. De Moor, and M. Fiedler, "Revealing the dark side of WebRTC statistics collected by Google Chrome," in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, June 2016.

[4] B. Garcia, F. Gortazar, L. Lopez-Fernandez, M. Gallego, and M. Paris, "WebRTC testing: challenges and practical solutions," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 36–42, 2017.

[5] B. García, M. Gallego, F. Gortázar, and A. Bertolino, "Understanding and estimating quality of experience in WebRTC applications," *Computing*, pp. 1–23, 2018.

[6] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman, "Performance evaluation of WebRTC-based video conferencing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 56–68, 2018.

[7] 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*. Springer, 2019, pp. 459–470.

[8] T. Spetebroot, S. Afra, N. Aguilera, D. Saucez, and C. Barakat, "From network-level measurements to expected quality of experience: the Skype use case," in *Measurements & Networking (M&N), 2015 IEEE International Workshop on*. IEEE, 2015, pp. 1–6.

[9] K. De Moor, S. Arndt, D. Ammar, J.-N. Voigt-Antons, A. Perkis, and P. E. Heegaard, "Exploring diverse measures for evaluating QoE in the context of WebRTC," in *Quality of Multimedia Experience (QoMEX), 2017 Ninth International Conference on*. IEEE, 2017, pp. 1–3.

[10] Y. Sulema, N. Amram, O. Aleshchenko, and O. Sivak, "Quality of experience estimation for WebRTC-based video streaming," in *European Wireless 2018; 24th European Wireless Conference*. VDE, 2018, pp. 1–6.

[11] S. Yan, Y. Guo, Y. Chen, and F. Xie, "Predicting freezing of WebRTC videos in wifi networks," in *International Conference on Ad Hoc Networks*. Springer, 2018, pp. 292–301.

[12] D. Ammar, K. D. Moor, and P. Heegaard, "An Experimental Platform for QoE Studies of WebRTC-based Multi-Party Video Communication," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 8, no. 2, pp. 89–94, 2018.

[13] Homepage of appear.in. [Online]. Available: https://appear.in

[14] H. Alvestrand and V. Singh, "Identifiers for WebRTC's Statistics API," W3C, W3C Working Draft, Feb. 2015.

[15] S. Winkler, A. Sharma, and D. McNally, "Perceptual video quality and blockiness metrics for multimedia streaming applications," in *Proceedings of the international symposium on wireless personal multimedia communications*, 2001, pp. 547–552.

[16] J. Greengrass, J. Evans, and A. C. Begen, "Not all packets are equal, part 2: The impact of network packet loss on video quality," *IEEE Internet Computing*, vol. 13, no. 2, pp. 74–82, 2009.

[17] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[18] Chromium, "The WebRTC project," https://chromium.googlesource.com/external/webrtc/.

[19] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey, "Extended RTP profile for real-time transport control protocol (RTCP)-based feedback (RTP/AVPF), IETF RFC 4585," IETF, Tech. Rep., 2006.

[20] S. Holmer, H. Lundin, G. Carlucci, L. De Cicco, and S. Mascolo, "A Google congestion control algorithm for real-time communication," in *IETF draft-ietf-rmcat-gcc-02*, 2016.

[21] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, Oct 2017.

[22] L. De Cicco, G. Carlucci, and S. Mascolo, "Experimental investigation of the Google congestion control for real-time flows," in *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*. ACM, 2013, pp. 21–26.

[23] C. Kilinc and K. Andersson, "A congestion avoidance mechanism for WebRTC interactive video sessions in LTE networks," *Wireless personal communications*, vol. 77, no. 4, pp. 2417–2443, 2014.

[24] S. Varma, *Internet Congestion Control*. Morgan Kaufmann, 2015.

[25] P. Reichl, S. Egger, R. Schatz, and A. D'Alconzo, "The logarithmic nature of QoE and the role of the Weber-Fechner Law in QoE assessment," in *Proceedings of IEEE International Conference on Communications, ICC 2010, Cape Town, South Africa, 23-27 May 2010*. IEEE, 2010, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/ICC.2010.5501894