TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Machine Learning for Network Traffic Monitoring and Analysis

Application to Internet QoE Assessment and Network Security

PhD THESIS

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

by

Sarah Anne Wassermann
Registration Number 11941823

to the Faculty of Electrical Engineering and Information Technology

at the TU Wien

Advisor: **Univ. Prof. Dr.-Ing. Tanja Zseby**, TU Vienna
Second advisor: **Dr. Pedro Casas**, AIT Austria

External reviewers:
Prof. Stefano Secci. Conservatoire des arts et métiers, France.
Prof. Anna Brunström. Karlstad University, Sweden.

Vienna, 6th April, 2022

Abstract

The Internet plays a crucial role in today's society. Indeed, numerous everyday tasks can now be accomplished online. For instance, it has become very easy to carry out a bank transfer via our computer or smartphone. While digitalization simplifies processes, it is very attractive for cybercriminals, as they can potentially access very sensitive and valuable user information, or even interconnected critical infrastructures. Protecting confidential data is now paramount and more challenging than ever before. We also increasingly rely on the Internet for entertainment. Whereas we used to watch movies on DVDs, we use today streaming services like Netflix. Statistics confirm this trend: according to Sandvine, video streaming made up 60% of the global Internet traffic in 2019. The COVID-19 pandemic has intensified this tendency: Netflix, YouTube, and Amazon even decided to reduce the default playback resolution to avoid a service breakdown during the lockdown. However, services need to deliver the best Quality of Experience (QoE) to avoid churn. The same is true for Internet service providers, whose goal is to operate their networks efficiently and to avoid severe QoE degradations. Monitoring their networks also becomes more difficult, as end-to-end encryption renders traditional Deep-Packet-Inspection techniques unreliable.

In this thesis, we leveraged machine learning (ML) for advanced network-traffic monitoring, with a specific focus on (*i*) the analysis of video and Web QoE, and (*ii*) the automatic detection of malware and network attacks, both relying on in-network and in-device measurements. All the proposed solutions were thoroughly evaluated on top of large and heterogeneous datasets. More precisely, our work revolved around the following topics:

Video QoE: we conceived two ML-based solutions to infer key QoE indicators (KQIs) for video-streaming services, the first one built on top of data gathered with YoMoApp, an Android application for session-based YouTube-QoE monitoring, the second one integrated into ViCrypt, a system for real-time sensing of relevant quality metrics of video streaming. ViCrypt infers the considered metrics every second while the user is watching a YouTube video, relying on in-network, encrypted-traffic measurements. To the best of our knowledge, this is the finest granularity so far used for quality inference in the context of encrypted traffic. With our two solutions, we achieved highly promising results in the context of video-KQI inference.

Web QoE: we built an AI-based framework to estimate the SpeedIndex of Web sessions and the levels of Web QoE from encrypted-network-traffic measurements. We showed that ML models did not generalize well across the considered devices (desktop, smartphone, tablet): a model trained on data coming from a single device yielded disappointing results when applied to data coming from other device types. To the best of our knowledge, we are the first to unveil the strong impact of the device type on the quality of Web-QoE inference. We additionally conceived multi-device, Web-QoE estimation models based on flow-level measurements, achieving high accuracy while being easily deployable.

Mobile-malware detection: we conceived BIGMOMAL – Big Data Analytics for Mobile Malware Detection –, an ML-based system for mobile-malware detection and application fingerprinting. BIGMOMAL exclusively relies on lightweight ML models and features extracted directly on the users' smartphone, while respecting their privacy. Through BIGMOMAL, we realized both tasks with high accuracy.

Stream-based active learning for detection of network attacks: we aimed at continuously tuning a supervised-learning model with limited availability of labeled data. We developed RAL – Reinforced stream-based Active Learning –, a framework based on stream-based active learning coupled with reinforcement learning to wisely choose the training points from which ML models can learn while still being able to infer crucial metrics with high accuracy. RAL outperformed the state of the art by increasing the estimation accuracy and lowering the number of necessary labeling queries. While we have applied RAL exclusively in network security problems in this thesis, it has been designed for any type of stream-based learning application.

In conclusion, we investigated highly relevant QoE and cybersecurity problems and proposed innovative data-driven solutions extending the state of the art. We advanced the broad domain of network-traffic monitoring and analysis by pushing forward the application of AI/ML to networking problems (AI4NETS), paving the way for a better Internet.

Kurzfassung

Das Internet spielt heute eine wichtige Rolle. Zahlreiche alltägliche Aufgaben können online erledigt werden. So ist es zum Beispiel sehr einfach geworden, eine Banküberweisung über den Rechner oder das Smartphone zu tätigen. Während die Digitalisierung vieles vereinfacht, ist sie für Cyberkriminelle sehr attraktiv, da sie potenziell auf sehr sensible Informationen oder sogar auf vernetzte kritische Infrastrukturen zugreifen können. Datenschutz ist bedeutender und eine größere Herausforderung als je zuvor. Auch für die Unterhaltung verlassen wir uns zunehmend auf das Internet: früher sahen wir uns Filme auf DVDs an, heute nutzen wir Streaming-Dienste wie Netflix. Statistiken bestätigen diesen Trend: Laut Sandvine machte Videostreaming 2019 60 % des weltweiten Internetverkehrs aus. COVID-19 hat diese Tendenz verstärkt: Netflix, YouTube und Amazon entschieden sogar, die Standard-Auflösung zu reduzieren, um einen Zusammenbruch während dem Lockdown zu vermeiden. Die Dienste müssen jedoch die beste Quality of Experience (QoE) liefern, um Abwanderung zu vermeiden. Das Gleiche gilt für Internetdienstanbieter, deren Ziel es ist, ihre Netze effizient zu betreiben und schwerwiegende QoE-Verschlechterungen zu vermeiden. Außerdem wird die Überwachung ihrer Netzwerke schwieriger, da die Verschlüsselung Deep Packet Inspection unzuverlässig macht.

In dieser Arbeit setzten wir maschinelles Lernen (ML) für die fortschrittliche Überwachung des Netzwerkverkehrs ein, mit besonderem Schwerpunkt auf (*i*) der Analyse der Video- und Web-QoE und (*ii*) der automatischen Erkennung von Malware und Netzwerkangriffen, die beide auf netzwerk- und geräteinternen Messungen beruhen. Alle vorgeschlagenen Lösungen wurden anhand von großen und heterogenen Datensätzen gründlich evaluiert. Genauer gesagt, drehte sich unsere Arbeit um die folgenden Themen:

Video-QoE: Wir haben zwei KI-Lösungen für die Inferenz wichtiger QoE-Indikatoren vorgeschlagen: eine basiert auf Daten von YoMoApp, eine Android-App für sitzungsbasierte YouTube-QoE-Erfassung, und eine integriert in ViCrypt, ein System für die Echtzeit-Erfassung relevanter Metriken des Videostreaming. ViCrypt ermittelt diese jede Sekunde wenn der Nutzer sich ein YouTube-Video ansieht, und nutzt dazu netzinterne, verschlüsselte Verkehrsmessungen. Dies ist unseres Wissens nach die feinste Granularität, die bisher für die Qualitätsinferenz im Kontext von verschlüsseltem Datenverkehr verwendet wurde. Mit beiden Systemen erzielten wir vielversprechende Ergebnisse.

Web-QoE: wir schätzten den SpeedIndex und das QoE-Niveau von Web-Sitzungen, indem wir ausschließlich verschlüsselte Netz-Level Eingaben nutzten. Wir haben gezeigt, dass ML-Modelle nicht gut über die analysierten Geräte (Rechner, Smartphone, Tablet) hinweg funktionierten, d.h. dass ein Modell, das auf Daten von einem einzigen Gerät trainiert wurde, enttäuschende Ergebnisse lieferte, wenn es auf Daten von anderen Geräten angewendet wurde. Unseres Wissens nach sind wir die ersten, die den starken Einfluss des Gerätetyps auf die Qualität der QoE-Inferenz zeigen. Außerdem haben wir Modelle erstellt, die Flow-Level- statt Paket-Level-Eingaben nutzen; sie lieferten sehr gute Ergebnisse und können gleichzeitig einfach eingesetzt werden.

Erkennung mobiler Malware: wir entwickelten BIGMOMAL – Big Data Analytics for Mobile Malware Detection –, ein System für die Erkennung von Malware und das Fingerprinting von Apps auf Android-Telefonen. BIGMOMAL nutzt ausschließlich einfache Modelle und Eingaben, die direkt auf dem Gerät extrahiert werden, unter Wahrung der Privatsphäre. BIGMOMAL ermöglicht es beides mit hoher Genauigkeit durchzuführen.

Datenstrom-basierendes aktives Lernen für die Erkennung von Netzwerkangriffen: Unser Ziel ist die kontinuierliche Optimierung eines überwachten Lernmodells bei begrenzter Verfügbarkeit von markierten Daten. Wir haben RAL – Reinforced Stream-based Active Learning – entwickelt, ein System, das aktives Lernen mit bestärktem Lernen nutzt, um die Trainingspunkte der ML-Modelle im Strom klug auszuwählen und noch in der Lage zu sein, Metriken mit hoher Genauigkeit zu schätzen. RAL übertraf den Stand der Technik, indem es die Schätzgenauigkeit erhöhte und die Anzahl der erforderlichen Datenabfragen verringerte. Im Rahmen dieser Arbeit haben wir RAL ausschließlich für Netzwerkangriffe verwendet, obwohl es für alle Arten von Datenströmen entwickelt wurde.

Zusammendfassend lässt sich sagen, dass wir wichtige QoE- und Sicherheitsprobleme erforscht und innovative datengesteuerte Lösungen vorgeschlagen haben, die den Stand der Technik erweitern. Wir haben den breiten Bereich der Überwachung und Analyse des Netzwerkverkehrs vorangebracht, indem wir die Anwendung von KI/ML auf Netzwerkprobleme (AI4NETS) vorangetrieben und so den Weg für ein besseres Internet geebnet haben.

Acknowledgments

With the submission of this thesis, a long journey comes to an end. Even though it has not always been easy, it has been a very enriching period of my life. Throughout the last couple of years, I have been shaped both as a researcher and as a human being by a handful of unforgettable people, to whom I want to express my sincerest gratitude.

First and foremost, I would like to thank Pedro Casas. I am immensely grateful to you. Since my first internship in Vienna in 2015, thanks to which I discovered the exciting world of research in networking, you have always believed in me, even in difficult times and when I was about to give up. Thanks to your commitment and support, I never felt alone when I faced a challenge, which was an invaluable help. You always had an open ear for all my concerns and doubts, and our collaborations and insightful discussions allowed me to grow in many aspects. You are not only an inspiring researcher, but also an exceptional human being. The completion of this PhD would not have been possible without you.

I am also very grateful to Tanja Zseby. Thank you so much for letting me do my PhD under your supervision. Your many constructive comments greatly contributed to the quality of this manuscript.

I also want to thank Michael Seufert. Working with you was a real pleasure and I learned a lot. Our interesting discussions gave me a better picture of the life of a young researcher. I also had a lot of fun when I had the opportunity to visit you in Würzburg.

Even though he was not directly involved in this thesis, I am extremely grateful to Marc, who largely contributed to my academic success. Throughout my high-school and university years, you helped me understand the world of mathematics, physics, and chemistry. You always answered my innumerable questions with a lot of patience. I will never be able to thank you enough for all the time and energy you sacrificed to support me.

I thank my partner Thibaut for his unconditional love and support. I can discuss with you about simply everything, and you always help me in any possible way. For example, you spent countless hours proofreading this manuscript, many more than I could have hoped for. All the adventures we are living together are very precious memories and make my life much more fulfilling. I could not imagine a more loving and caring partner.

A special thank you goes to my mother, who supported me throughout my whole life as best as she could. You greatly helped me when I was going through the hardest times of my life. During my studies, you suffered with me during the exam periods and made sure that I did not have to worry about anything else than my academic career.

The next lines are heartbreaking to write. I want to express my deepest gratitude to my best friend, Caroline, even though she will unfortunately never be able to read those acknowledgments. Even though you left us almost two years ago, it is still impossible to describe the pain. I want to thank you for all the moments we spent together, the countless hours we spent together laughing and talking about our doubts and fears. You were always trying to help me, even though you were facing your own problems. I unfortunately never thanked you enough for being such a wonderful friend and for your trust.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ABR	adaptive bitrate
ACR	absolute category rating
ADA	AdaBoost
AFT	above the fold time
AI4NETS	AI for networks
AL	active learning
AUC	area under the ROC curve
BI	ByteIndex
BIGMOMAL	big data analytics for mobile malware detection
CDF	cumulative distribution function
CSS	cascading stylesheet
CSSOM	CSS object model
DASH	dynamic adaptive streaming over HTTP
DBSCAN	density-based spatial clustering of applications with noise
DNS	domain name service
DOM	document object model
DPI	deep packet inspection
DT	decision tree
ERT	extremely randomized tree
FCP	first contentful paint

HAS	HTTP adaptive streaming
HEVC	high efficiency video coding
HLS	HTTP live streaming
HTML	hypertext markup language
HTTP	hypertext transfer protocol
IAT	inter-arrival time
ICMP	Internet control message protocol
ISO	isolation forest
ISP	Internet service provider
JS	JavaScript
kNN	k-nearest neighbors
KPI	key performance indicator
KQI	key QoE indicator
LOF	local outlier factor
MAE	mean absolute error
ML	machine learning
MOS	mean opinion score
MPEG	moving picture experts group
MRE	mean relative error
NAT	network address translation
NB	naive Bayes
NN	neural network
OI	ObjectIndex
PCA	principal component analysis
PHT	Page-Hinkley test
PLCC	Pearson linear correlation coefficient
PLT	page load time

QoE	quality of experience
QoS	quality of service
QUIC	quick UDP Internet connections
RAL	reinforced stream-based active learning
RAT	radio access technology
RDA	rate determination algorithm
RF	random forest
RL	reinforcement learning
RMSE	root mean squared error
ROC	receiver operating characteristic
RS	random sampling
RUMSI	real user monitoring SpeedIndex
RVU	randomized variable uncertainty
SI	SpeedIndex
SMOTE	synthetic minority over-sampling technique
SNMP	simple network management protocol
SVM	support vector machine
TCP	transmission control protocol
TTFB	time to first byte
TTFP	time to first paint
TTI	time to interactive
TTL	time to live
UDP	user datagram protocol
URI	uniform resource identifier
URL	uniform resource locator
VCR	visual-completion ratio
WPT	WebPageTest
XGB	XGBoost

CHAPTER 1

Introduction

As our lives become more and more dependent on the Internet, two factors play a crucial role in the online world, namely Quality of Experience (QoE) and security.

Indeed, we are increasingly frustrated when poor Internet performance prevents us from accomplishing evermore important online activities. The QoE when accessing the Internet is thus a key factor for today's society. Poor Internet QoE is annoying, in particular for most Internet users who are not tech savvy and hence cannot diagnose – let alone fix – problems by themselves. The interactions of Internet services with the networks they use have become increasingly difficult to predict, which hampers the attempts to diagnose QoE, e.g. due to proliferation of proxies and caches at the network core, of home wireless, and of 3G/4G/5G access. It is hard even for networking experts to fully diagnose and fix problems. When users get frustrated with degraded QoE, they may stop using services, and companies get the blame (and possibly lose money). Despite the relevance of QoE, we still face limitations to correctly infer and assess QoE, especially in real operational deployments. Diagnosing the causes for poor QoE related to the network and to application-layer decisions remains an open challenge. One factor which makes QoE monitoring and modeling especially difficult in today's Internet is end-to-end encryption. Indeed, encryption has rendered previous Deep-Packet-Inspection (DPI) approaches highly inaccurate or even infeasible. Figure 1.1 illustrates this issue for the monitoring of YouTube QoE: while encrypted network-layer features such as packet sizes, inter-arrival times, and throughput can still be measured by the ISPs, application-layer information such as quality levels and stallings can only be monitored on the users' devices.

Apart from entertainment, we use the Internet for essential everyday tasks. Indeed, we might want to complete critical online activities, such as bank transfers or organizing medical appointments, and these require us to enter personal information. This is why personal computers, and especially mobile devices, become more and more the target of malicious attacks that aim at stealing the users' private information and using

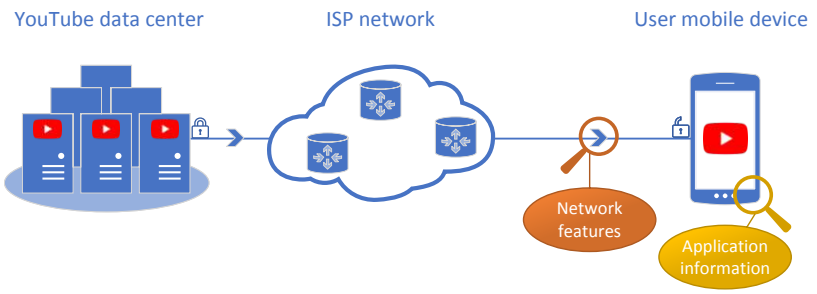


Figure 1.1: YouTube-QoE monitoring: the metrics of interest need to be measured at different locations in the network due to end-to-end encryption.

it for bad-natured activities. Smart devices are not only an appealing target because of their popularity, but also because they incorporate a very large amount of sensitive information, even more than a personal computer [4]. For instance, the included sensors can track the user's current location and physical activities. As technology evolves, cybercriminals are enhancing their attack methods, tools, and techniques to exploit individuals and organizations. We are facing a ramping-up cyberarms race, where attackers are constantly finding clever ways to bypass security systems and to reach their targets.

In this thesis, we leveraged machine learning to tackle the problem of QoE monitoring and modeling as well as of cybersecurity in dynamic, heterogeneous, and encrypted environments.

1.1 Overall Background in AI for Networks (AI4NETS)

In the last years, the popularity of artificial intelligence (AI) – and the one of machine learning (ML) as an approach to AI – has significantly increased due to its outstanding performance in a wide range of domains such as video, audio, and natural language processing. Recently, AI has also become more and more important in games thanks to recent developments in deep neural networks and reinforcement learning [5, 6].

Unfortunately, when it comes to AI4NETS, the picture looks less promising. The application of AI concepts and ML approaches to networking problems has today more than two decades of existence, with the first steps taken back in 1998, with the introduction of the concept of cognitive networks [7] by researchers at KTH Sweden. This paradigm characterizes a network with cognitive capabilities which could learn from past observations and behaviors, to better adapt to end-to-end requirements. Cognitive networks have been strongly refurbished along time, referring to them as self-organizing networks, self-aware networks, self-driving networks, and intelligent networks, for instance. This kind

of network served as a motivation for a large number of papers applying ML principles to current networking problems, such as traffic prediction, traffic classification, traffic routing, congestion control, network resource management, network security, anomaly detection, and Quality of Service (QoS) and QoE management [8, 9, 10, 11, 12, 13, 14]. A common trend we find in the existing literature is that there is a systematic lack of analysis on the multiple aspects which could lead to eventually re-use and reproduce, generalize, or even apply the obtained results in real deployments, for the most part of those papers. In a nutshell, most of what we have in ML for networking has been about grabbing one particular algorithm and testing it on some particular networking dataset – hopefully large, but in reality, of limited size and most probably of limited representativeness of the studied problems.

Nevertheless, we are seeing today multiple papers applying more modern flavors of ML to important networking problems. Some examples are deep learning [15, 16, 17], transfer learning [18], and explainable AI [19] for network security, and deep reinforcement learning for network management [20, 21, 22]. However, the speed of adoption of AI-based solutions to current networking problems is extremely slow and we are still facing a striking gap between the academic research efforts and the actual deployments of such systems in operational environments.

One might wonder why the development of AI in the domain of networking is stalling as compared to other applications. There are a couple of reasons worth mentioning:

Data complexity and diversity: the data coming from networks is very heterogeneous, as they are made up of a wide range of diverse technologies, applications, services, devices, and end users. It is thus very difficult to learn something out of the complex interactions between all those components. Eric Schmidt, former CEO of Google, rightfully said "*the Internet is the first thing that humanity has built that humanity does not understand, the largest experiment in anarchy that we have ever had*". The domains in which AI is very popular mostly rely on structured, more predictable, and easy to understand data, as for instance images. With the ever-growing number of devices connected to large-scale networks and the continuous emergence of new applications and services running on them, the diversity and unpredictability of networking data is constantly increasing.

Data dynamics: another challenge faced by networking-data analysts is the fact that the data is dynamic and comes most often in the form of data streams, which need to be rapidly and continuously processed. Moreover, these streams are subject to concept drifts, i.e. changes in the underlying statistical properties, making their accurate processing and estimation even more difficult.

Lack of learning generalization: due to the complexity and dynamics of the data, it is very difficult to build ML models which generalize well to other environments, i.e. environments including data different from the training data. In particular, this means that models built and calibrated based on a certain network have a high

risk of not working well in other networks, which is problematic in case a model should be deployed in an operational environment.

Lack of ground truth: the methods of supervised learning, which are for now the dominant for networking problems, require a large amount of labeled training data. However, most networking datasets including "in the wild" data are unlabeled, as labeling complex and dynamic data is very costly and error-prone. Another factor which makes building representative datasets difficult is the sensitive nature of data points, as we need to preserve the users' privacy. Finally, given the scale of Internet networks, their massive volumes of data, and the multiplicity of operational conditions, building such a representative dataset is a daunting task.

High costs of errors and lack of performance bounds: deploying an AI-based system in operational environments comes with its fair share of risks, as the deployments can result in costly errors. This is especially risky for critical applications such as the medical field or cybersecurity. Very often, Internet service providers (ISPs) and network-equipment vendors are not willing to bear these risks. Moreover, being data-driven by nature, and prone to outliers, it is extremely challenging to provide tight performance bounds on trained ML models. Robust learning is paramount for networking. Finally, most ML models lack transparency, which limits their application even further. If ISPs cannot understand why a particular decision was taken by the model, they will not trust it and therefore not use it.

Learning in adversarial settings: the world of networking presents a lot of adversarial examples. An obvious example is security, where an arms race is evident. However, we also face this challenge for other areas such as traffic-identification and classification tasks, because user applications do not want to be tracked by intermediate entities, and therefore obfuscate and dynamically modify their functioning to bypass monitoring and avoid traffic-engineering policies.

1.2 Contributions of the Thesis

In the context of this thesis, we aimed at contributing to AI4NETS. We leveraged machine-learning techniques to build AI-based systems for QoE monitoring & modeling and cybersecurity in dynamic, heterogeneous, and encrypted environments. In particular, the thesis revolves around three main axes:

1. **ML-based QoE modeling for video-streaming services:** *can we accurately infer video QoE from network-layer traffic despite end-to-end encryption*? To answer this question, we investigated machine-learning approaches to infer key QoE indicators (KQIs) from encrypted traffic, by exclusively relying on network-layer features. We looked into ML-based video-QoE monitoring and modeling both on mobile devices as well as on desktop machines. We explored two different methods for QoE monitoring:

a) Session-based QoE monitoring on mobile devices, i.e. we inferred KQIs after the user watched a video within a dedicated application from network data easily accessible through the Android API.

b) Stream-based QoE estimation on desktop, i.e. we estimated KQIs for each 1-second time slot while the user was watching a YouTube video, which is, to the best of our knowledge, by now the finest time granularity for real-time estimation of quality metrics from encrypted traffic.

We obtained very encouraging results for both scenarios.

2. **ML-based Web-QoE monitoring:** *can we accurately infer Web QoE from network measurements? Are models trained on data coming from a specific device type applicable in a heterogeneous ecosystem?* We applied ML methods to infer key Web-QoE metrics from network data on different types of devices, more precisely on desktop machines, smartphones, and tablets. In particular, we focused on the Real User Monitoring (RUM) SpeedIndex, a passive approximation of the widely used SpeedIndex metric. In the context of this work, we tackled Web-QoE estimation both as a regression and a classification task. Our results showed that ML models trained on data coming exclusively from a single device type did not generalize well to other devices. However, we found that models learning from multi-device measurements yielded very promising inference accuracy. Besides ML models based on packet-level features, we also explored ML models relying on flow-level features, which are much more lightweight, and found that both types of models performed similarly well.

3. **ML-based detection of security breaches:** *can we rely on machine learning for security-related tasks on smartphones? In particular, how can we handle scarce resources to uncover malicious application behavior and network attacks?* To investigate these questions, our research efforts for cybersecurity were two-fold. First, we studied the problem of malware detection and application fingerprinting in smartphones, using supervised machine learning. We obtained promising results with lightweight decision trees. Next, we tackled the problem of data labeling. As mentioned in Section 1.1, ground-truth data is a scarce resource and obtaining accurately labeled data is a tedious task, especially if humans are involved. When facing a fast incoming data stream, one might want to avoid having to label each and every sample, but only the most beneficial ones (the definition of "beneficial" can of course vary from one use case to another). One particular scenario where this challenge appears is security: labeling a data point as malicious or benign can be costly and time consuming; querying only the most interesting samples greatly improves the whole process. We developed a framework based on stream-based active learning coupled with reinforcement learning to wisely choose the training points from which ML models can learn while still being able to infer crucial metrics with high accuracy.

1.3 Outline of the Thesis

The remainder of this thesis is divided into five chapters. More specifically, it is organized as follows:

Chapter 2 "Background": this chapter explains the basic notions underlying the topics that are tackled in this thesis, namely video streaming, Web browsing, network measurements, and machine learning.

Chapter 3 "Video-QoE Monitoring and Analysis": this chapter presents the AI-based frameworks we developed for video-QoE monitoring and modeling, in particular built on top of YoMoApp data [23] for session-based KQI inference on mobile devices and integrated into ViCrypt [24] for real-time KQI estimation on desktop machines. Both frameworks rely exclusively on lightweight network-layer features. The chapter also describes the results we obtained for the two systems.

Chapter 4 "Web-QoE Monitoring and Analysis": this chapter details our ML-based Web-QoE-inference methodology. It explains how we collected our data and its characteristics. Moreover, this chapter gives insights into the different ML models we built to tackle the Web-QoE-estimation challenge and into the corresponding results. For instance, we show the lack of cross-device generalization of models and present a promising solution.

Chapter 5 "(Adaptive) Detection of Network Attacks": this chapter introduces BIGMOMAL and RAL. With BIGMOMAL – Big Data Analytics for Mobile Malware Detection –, we were able to perform application fingerprinting and malware detection on Android phones with high accuracy. BIGMOMAL exclusively relies on simple ML models and on lightweight features extracted directly on the users' phone, in respect of their privacy. RAL – Reinforced stream-based Active Learning –, is a system coupling reinforcement learning with stream-based active learning. RAL allows ML models to wisely choose the most informative samples from which they should learn. We show that the models still achieved high inference accuracy for the considered estimation targets even though they were trained on a reduced dataset. In this thesis, we applied RAL on security-related problems, even though the system has been designed to work with any type of data coming in the form of streams.

Chapter 6 "Conclusions and Perspectives": in this chapter, we conclude this thesis and discuss perspectives and potential future work.

CHAPTER 2

Background

This thesis revolves around the interplay of video streaming, Web, and machine learning. To ease the understanding of this manuscript, we provide background knowledge about these topics in this chapter.

In Section 2.1, we explain how adaptive video streaming works and how to assess its quality. In Section 2.2, we describe the basic functioning of a Web browser and the most important performance-evaluation metrics. In Section 2.3, we review how to gather network statistics. Lastly, in Section 2.4, we present an overview about ML concepts and algorithms.

2.1 Video Streaming

In this section, we introduce the basic notions of video streaming. We can distinguish between two types of streaming: progressive video streaming and adaptive video streaming. For the sake of completeness, we briefly explain progressive streaming, but mainly focus on adaptive streaming, as the video services we investigate in this thesis rely on the adaptive paradigm.

2.1.1 Progressive Streaming

Progressive streaming is a very simple approach to deliver a video to the user. Indeed, the video is streamed as single large file over the Internet. This implies that every user, regardless of her Internet connection and her device, downloads the same file from the video server. As a consequence, it might happen that the user is not satisfied with her video experience because of connection problems while downloading a too large file or because the video looks unnatural as it needs to be stretched/squashed to fit the device screen, for instance.

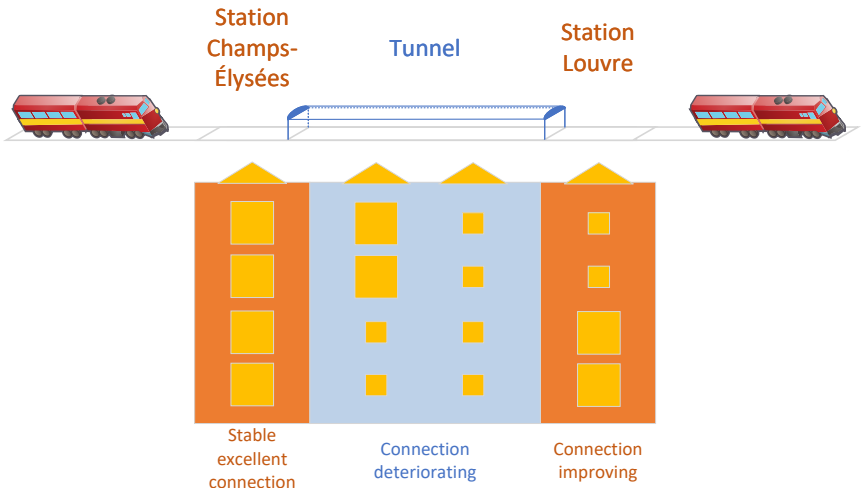


Figure 2.1: Example of HAS; the chunks are represented by the yellow blocks. When riding the metro, the user's Internet connection usually deteriorates. As soon as connection issues appear, HAS adapts the streaming behavior by starting to send smaller, lower-quality video chunks. The video quality progressively improves as soon as the connection gets back to normal again.

2.1.2 HTTP Adaptive Streaming

With HTTP Adaptive Streaming (HAS), the streaming becomes, as its name suggests, adaptive. In particular, in contrary to progressive streaming, HAS adapts the bitrate to the connection conditions of the user by, for instance, transferring lower-quality video chunks in case of connectivity issues. Figure 2.1 illustrates the purpose of HAS with an example. The main goal of HAS is to optimize the user's QoE.

Functioning

HAS requires the video provider to create multiple video files, each containing the video in a different quality. Moreover, each of these videos will be divided into segments having a fixed duration – typically between one and ten seconds. This allows the client to request segments of different quality levels depending on her Internet connection.

To initiate a video session, the user first downloads a manifest file from the server. This file contains information about the video and lists all its segments, the different quality levels they are available in, and their URIs. After this first step, the user's video player determines with the help of a Rate Determination Algorithm (RDA) the quality of the segments to download, based on multiple factors such as buffer level, state of the Internet connection, or CPU load. Thanks to the different quality levels of the video

and its segmentation, the RDA can easily adapt its video-chunk requests and avoid bad user QoE.

The two most commonly used HAS implementations are MPEG Dynamic Adaptive Streaming over HTTP (DASH) [25] and HLS [26]. Both approaches are very similar, but do present some differences.

HLS has originally been developed by Apple exclusively for the iPhone. However, almost every device is able to use this protocol today. HLS only plays videos encoded with the H.264 or HEVC/H.265 codecs.

DASH is the most recent competitor of HLS. As opposed to Apple's solution, DASH is video-codec independent. It also allows the insertion of advertisements. Major video services such as YouTube and Netflix rely on this protocol.

2.1.3 Video-Quality Assessment

There exist several ways to assess video quality. In this subsection, we briefly discuss one metric, namely the ACR Mean Opinion Score (MOS) [27], and one standard inference model for the MOS, the ITU-T P.1203 model [28].

The MOS score is a quality metric which was initially designed for the telephony domain. The standard 5-level score ranges from bad (MOS = 1) to excellent (MOS = 5). As it is a very intuitive score and easy to indicate (one mouse click is enough), it is a commonly used evaluation metric when it comes to quality studies involving humans. Today, more and more quality-evaluation algorithms try to map video statistics to a MOS score.

The ITU-T P.1203 model is a parametric model for audiovisual quality assessment of adaptive video streaming. It is the first standard for inferring the Quality of Experience of HTTP Adaptive Streaming services. It is composed of three different modules: the audio-quality-estimation module, video-quality-estimation module, and the quality-integration module. It outputs a series of quality metrics, such as audio/video quality per sampling interval (one second), and also a final video-session quality score, which is a mapping of all the input information to a MOS score.

2.2 Web Browsing

In this section, we provide an introduction to the concepts underlying Web browsing. The content displayed by browsers on the screen can be divided into two parts:

- a static part, described in the HTML file downloaded by the browser, that uses external resources like CSS for the style and images. The time needed to complete this part of rendering is mostly determined by network conditions.
- a dynamic part, obtained by executing JavaScript (JS) code on the client side. Its execution speed is largely determined by the characteristics of the processor.

2.2.1 Loading of a Webpage

We now explain the generic functioning of a modern Web browser by detailing how a Webpage is displayed on a screen, starting with the query of the user.

When the user enters the URL of the Webpage to show, the browser must first retrieve the HTML content of the page. It does so using the HTTP protocol (version 1.1 was predominant in the last few years, but versions 2 and 3 become more and more common) once the DNS record is resolved. Some Websites use many external resources and can indicate, at the beginning of the HTML page, which domains should also be resolved by DNS as soon as possible (as opposed to resolving when it becomes necessary), in order to lower the total latency for loading the site.

While the HTML content is being downloaded, the browser can fetch resources that are needed for the page (like stylesheets and images), in parallel (i.e. using several HTTP connections). As the resources are being loaded, the browser can start rendering the page and updating the view when new resources arrive or when changes are requested by the JavaScript code. Finally, when all the elements of the page are completely loaded, the browser fires the JavaScript `onLoad` event.

2.2.2 Rendering of a Webpage

To render an HTML page, the browser builds the corresponding DOM tree, a representation that is amenable to many operations, notably alterations via JavaScript code. This DOM tree only represents the data to render on a screen and has no notion of style. For instance, the DOM tree indicates that the page contains a title and a table.

In parallel to the DOM creation, the browser loads the style information conveyed through CSS. It is stored in a data structure analogous to the DOM, called the CSSOM. The CSSOM tree only represents style, independently of the actual contents of the Webpage. For instance, the CSSOM memorizes the space between two paragraphs.

Once the DOM and CSSOM are built, styles from the CSSOM are applied onto the DOM to determine how each element should be displayed. The resulting tree is called the render tree; compared to the DOM tree, the render tree has information about colors and dimensions.

Following this annotation step, the page is laid out: each element of the render tree is given exact coordinates in the viewport, which corresponds to the way the page is rendered on a screen of unlimited length (depending on the quantity of content to show).

Finally, a part of this viewport is painted on the screen, depending on the scrolling level within the page.

The complete process is illustrated in Figure 2.2.

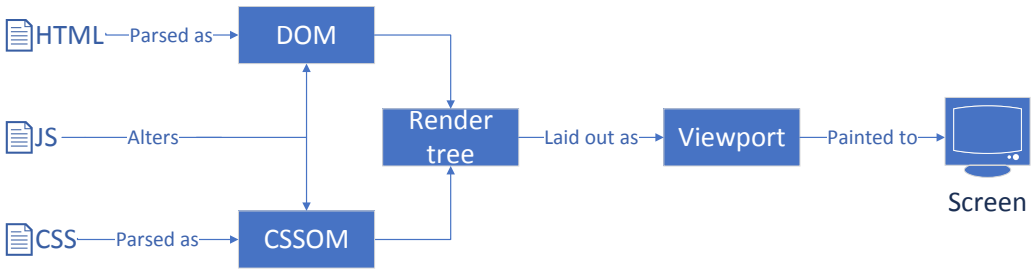


Figure 2.2: Page-rendering in a browser.

2.2.3 Web-QoE Assessment

In order to assess the user experience on the Web, we can use several performance metrics. We briefly describe a couple of the most commonly used ones here:

Time to First Byte (TTFB): this is a metric commonly used to assess the responsiveness of a Website by checking how fast the server responds. In particular, the TTFB indicates the time between the browser requesting a page and receiving the first byte of the response.

Time to First Paint (TTFP): this metric is very similar to TTFB. It measures the time until the first pixel is painted on the screen.

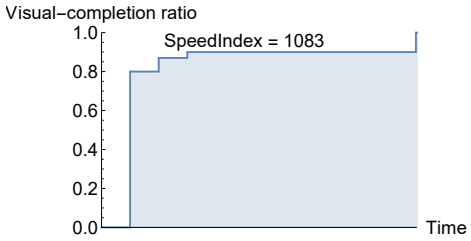
First Contentful Paint (FCP): it represents the time until the browser renders the first element of the DOM tree.

Page Load Time (PLT): it indicates the time needed to fully load the Webpage (i.e. until the `onLoad` event is fired).

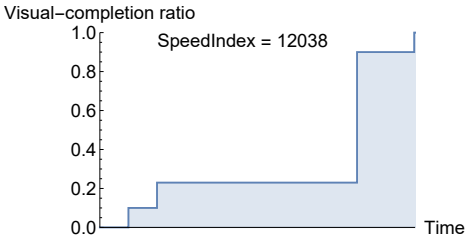
Above the Fold Time (AFT): a major drawback of the PLT is that it considers the whole Webpage, even though the user only sees a fraction of it on the screen (and needs to scroll to see the remaining parts). Therefore, the AFT was introduced, which indicates the time required to fully load the first sight encountered by the user during her visit, i.e. without scrolling. The term "above the fold" comes from the newspaper domain and designates the upper half of the front page containing the top stories.

Time to Interactive (TTI): the time it takes the Webpage to become fully interactive.

SpeedIndex (SI): this integral metric measures how quickly content is visually displayed while the page is loaded, by processing a video capture of the screen. The SI is therefore particularly interesting for evaluating the perception of speed. Mathematically, the SI can be expressed as $\int_0^{\texttt{onLoad}} [1 - \mathrm{VCR}(t)]\ \mathrm{d}t$ where VCR represents the visual-completion ratio over time. Figure 2.3 shows an example of both good



(a) Good SpeedIndex (optimized Website).



(b) Bad SpeedIndex (non-optimized Website).

Figure 2.3: SpeedIndex illustration. For better user experience, visual elements should load from the first seconds on.

and bad SpeedIndexes. It illustrates that it is better to have a Website whose elements are displayed from the first second on.

Real User Monitoring SpeedIndex (RUMSI): the RUMSI is a passive approximation to the SI proposed in [29], and it is computed from the analysis of Webpage-resource timings.

ObjectIndex (OI): this metric follows the same logic as the SI and has been introduced in [30]. It represents how quickly objects are downloaded.

ByteIndex (BI): another metric which has been proposed in [30], and similar to SI and OI. It shows how quickly the bytes are downloaded.

In Chapter 4, we mostly focus on RUMSI, but also analyze TTFP and PLT for different devices.

2.3 Network Measurements

In this thesis, we analyze a significant amount of data coming from the network. In this section, we therefore explain how we can actually measure the network and extract relevant information about its state. There exist two kinds of measurements, namely active and passive ones:

- By doing active measurements, we inject packets into the network based on which we can derive network performance. This has a negative side effect, as we may disturb the network traffic with these additional packets.
- By doing passive measurements, we monitor the network without injecting any additional traffic. This type of monitoring is usually done by including some kind of intelligence into the network devices to enable them to identify and record the packets flowing through them. With passive measurements, users can monitor a single point in the network, which can be very useful for debugging purposes.

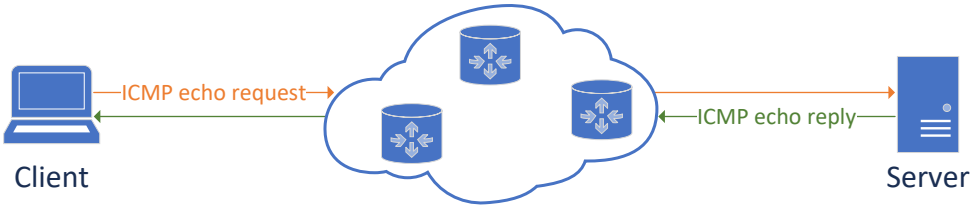


Figure 2.4: Ping overview.

We discuss both types for the sake of completeness, even though our work exclusively relies on passive measurements.

2.3.1 Active Measurements

We briefly present the two most commonly used active-measurement tools, ping and traceroute.

Ping

Ping [31] is a powerful tool built by Mike Muuss in 1983. Its purposes are twofold: first of all, it checks whether a server is reachable, and, if so, it measures the round-trip time (RTT) for messages sent from the client (i.e. the machine on which ping is executed) to the server. To do so, ping relies on the Internet Control Message Protocol (ICMP) [32]: it sends ICMP echo request packets (ping packets) towards the server and waits for an echo reply (pong packets). This waiting time is the RTT. If the waiting time is higher than a given threshold, ping considers the server as unreachable. Moreover, ping provides additional information such as packet loss, the mean RTT, and errors that occurred during its execution. The general working scheme of this tool is depicted in Figure 2.4.

Traceroute

Traceroute [33] is a tool allowing to infer a route from a client to a server. It was designed by Van Jacobson in 1989. Traceroute relies on UDP packets with a UDP destination port number that is likely unused (i.e. no process is listening on this port) and whose destination address is the one of the server. This tool makes use of the Time To Live (TTL) field and the IP forwarding mechanism in an ingenious way.

The basic idea behind traceroute is to discover the routers on the path by receiving ICMP time exceeded messages. Traceroute initially sends a UDP packet with a TTL equal to one and iteratively increments this value until the destination has been reached

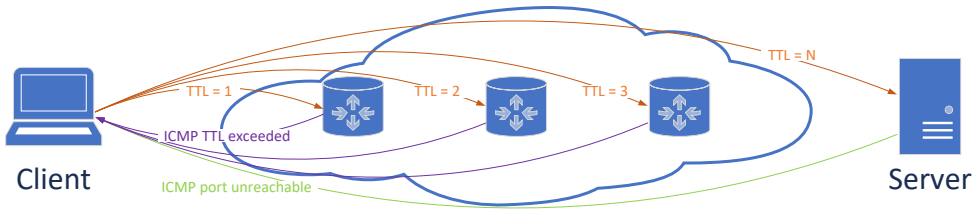


Figure 2.5: Traceroute overview.

(or the TTL has exceeded a value defined by the user). The general working scheme of traceroute is depicted in Figure 2.5.

In a nutshell, when the Nth packet arrives at the Nth router, this router observes that the TTL has expired. According to the forwarding rules of the IP protocol, this router discards the datagram, but sends an ICMP time exceeded message back to the client, which can then use the received packet to extract the IP address of the given router. The TTL of a datagram can thus be considered as its "life bar": each encountered router decreases it by one and, as soon as it is equal to zero, the packet "dies" (i.e. is not forwarded anymore). This TTL mechanism therefore prevents a cycling packet from wasting too many resources which could be useful for others. We have to point out that not every router sends an ICMP message back to the source. In this case, a timeout will be triggered and this hop will be indicated by a * instead of an IP address. When the target has been reached, the source receives an ICMP port unreachable message instead of a time exceeded message.

In addition, traceroute reports for each hop the RTT. Indeed, a timer is started when a new packet is sent, which allows traceroute to determine the time elapsed between the transmission of the UDP packet and the reception of the time exceeded/port unreachable message.

2.3.2 Passive Measurements

We briefly introduce two approaches to carry out passive network measurements. In particular, we explain the tcpdump and Wireshark tools as well as the Simple Network Management Protocol (SNMP).

Tcpdump and Wireshark

Tcpdump and Wireshark are packet-capturing tools. They listen to all traffic on a given network interface and have very similar functionalities. The user has the possibility to apply filters on the incoming and outgoing packets, such as specifying the source IP or recording only TCP flows. Only the matching packets are picked for potential

further analysis. The main difference between the two software packages is the interface: tcpdump only offers a command-line interface, while Wireshark also provides a graphical one. Moreover, according to [34], Wireshark captures high-speed connections better than tcpdump, which is very important in today's ever faster Internet. In this thesis, we rely on these kinds of tools for our measurements.

A similar tool is TStat[35], which does not simply capture packets, but gives insights into the network state by computing statistics on the network and transport levels.

Simple Network Management Protocol

SNMP [36] is a protocol whose goal is to simplify management of network equipment. One of its main features is the ability to gather statistics from switches, routers, firewalls, and other network functions, which can be periodically retrieved on a centralized server, the NMS (network management server). The available information depends on the manufacturer of each device, even though some standardization happened for commonly monitored fields. Typically measured values are the number of TCP flows witnessed by the device, the number of undeliverable UDP datagrams, or the failure of a specific network interface.

2.4 Machine Learning

In the context of this thesis, we heavily rely on machine learning (ML) to infer our metrics of interests. We therefore introduce the basic ML concepts and algorithms in this section. In particular, we will introduce several approaches, namely supervised, unsupervised, active, and reinforcement learning. As we barely use the unsupervised paradigm in our research, we mostly focus on the other ones.

2.4.1 Supervised Learning

Supervised ML is the most commonly used paradigm. The goal is to determine a function that maps an input database, consisting of labeled data points, to an approximation of the output. More formally, from a labeled learning database $\{(x_i, y_i)|i = 1 \dots N\}$ with $x_i \in X$ and $y_i \in Y$, we want to conceive a function $f : X \to Y$ that minimizes the expectation of some loss function.

We differentiate between classification and regression tasks. While we infer discrete class labels when carrying out a classification task (for instance, an animal shown on a picture), we estimate continuous quantities in the context of regression (the height of a person, for example).

Algorithms

There exist a multitude of algorithms generating models that approximate the output of learning databases. We briefly present the most well-known ones:

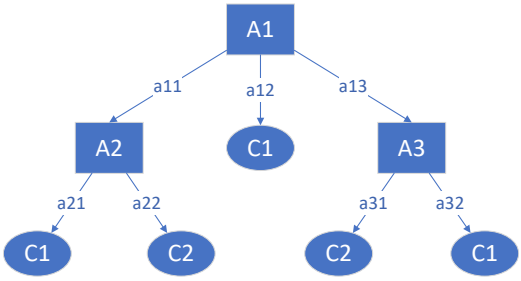


Figure 2.6: Decision-tree example. A1, A2, and A3 are selected features, and a_{ij} are the corresponding possible values of those features. C1 and C2 are the available classes.

Linear regression: a relatively simple statistical model. The inferred value $\hat{y}$ is a linear combination of the input-feature vector X containing F features: $\hat{y} = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_F x_F$, where θ is the vector of parameters which we need to determine to approximate the ground truth in the best possible way. Linear regression can be easily adapted to classification problems. For instance, we can output $\hat{y} = 1$ if $\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_F x_F > 0$ and $\hat{y} = 0$ otherwise.

Decision tree [37]: a hierarchical structure with one feature-based decision rule per node. In other words, at each node, the model asks a question about one specific feature, and the answer decides the next question, until a leaf is reached. The feature is selected using the notion of impurity, which is a measure of the class dispersion (if all samples have the same class, the impurity is minimum). In particular, the feature for each node is decided by the level of impurity reduction that is achieved splitting the dataset based on this feature. One very frequently used impurity metric is the Gini index [38]. For continuous attributes, the split is made at the cut-point value, also determined to reach the largest decrease in impurity. The content of the leaf node determines the inferred output. Figure 2.6 illustrates an example.

***k*-nearest neighbors [39]:** an intuitive algorithm which computes the output of the sample to label from the labels of the k closest training examples in the feature space (k being a hyperparameter). An example is shown in Figure 2.7.

Naive Bayes [40]: a probabilistic classification algorithm which computes the most likely output using the Bayes theorem assuming that all features are independent.

Support-vector machine [41]: an algorithm which computes a maximum-margin hyperplane to split the training samples into two classes. As SVMs are solely binary classifiers, the multiclass implementation is done via the one-versus-all paradigm, i.e., an SVM is built for each class and tests whether a sample belongs to that particular class or another one. A regression version also exists, which is called support-vector regression. An example of an SVM is depicted in Figure 2.8.

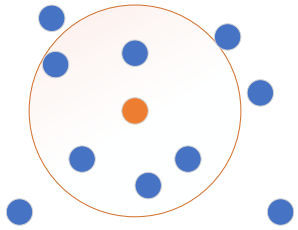


Figure 2.7: k-nearest-neighbors example with $k = 5$. The labels of the 5 samples closest to the orange point to consider (the 5 data points in the red circle) are combined to compute the label of the point under consideration.

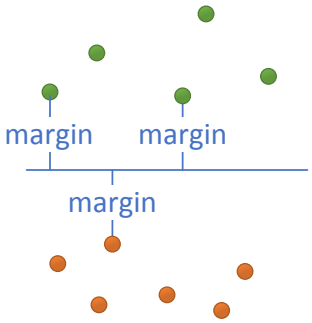


Figure 2.8: SVM example.

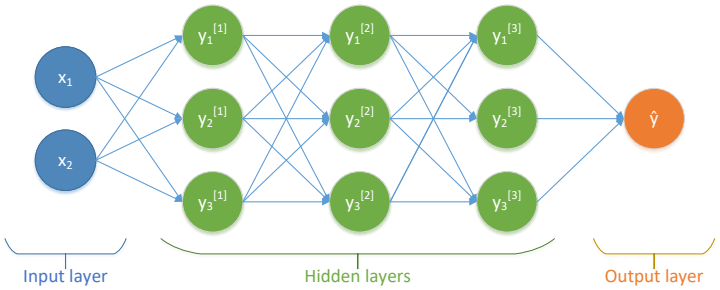


Figure 2.9: Neural-network example.

Neural network [42]: an algorithm which is inspired by the way the human brain works. The input data is transiting through a series of connected neurons until it reaches the output layer. Each neuron computes a non-linear function of its inputs. A network is composed of an input layer (the input features of the sample), the output layer (the estimated output), and optionally one or several hidden layers between the input and output layer. Each layer contains one ore several neurons. An example topology is shown in Figure 2.9.

Now that we have covered the most basic algorithms, we discuss more complex approaches. Indeed, different models can be combined using different approaches, such as averaging, boosting, or stacking. These techniques are called ensemble methods, as more than one model is involved:

Averaging [43]: techniques which build several models independently, potentially in parallel. They obtain their outputs by averaging the outputs of the different models in case of regression, or by carrying out majority voting in classification tasks. There exist multiple averaging algorithms:

Bagging [44]: bagging stands for Bootstrap Aggregating. In the context of this approach, we build multiple models derived from the same algorithm on top of bootstraps, i.e. samples drawn from the original dataset with replacement.

Random forest [45]: an ensemble method which builds a set of decision trees, each using only a randomly selected subset of samples (i.e. bootstraps) and a random subset S of features for determining the best split at each node; the cut point for each candidate feature in S is chosen optimally, as in a simple decision tree.

Ensemble of randomized trees [46]: an ensemble method similar to random forests, but with two main differences: (i) each extremely randomized tree is trained on the whole training set, (ii) at each node, similar to a random forest, the best feature is selected in a randomly chosen subset S, but the cut point for each candidate feature in S is determined randomly instead of being optimal.

Boosting [47]: in the context of boosting, we build good models based on poor ones. As opposed to averaging, the models are built sequentially and not in parallel, as the creation of a new model relies on information of the previous one. Similarly to the averaging methods, the inferences of the all the models are aggregated to obtain the output, however, the different learners' outputs are weighted. AdaBoost, Gradient Boosting, and XGBoost are well-known boosting algorithms:

AdaBoost [48]: AdaBoost stands for Adaptive Boosting. It starts by fitting a very simple and weak model on the training set. It then sequentially creates new models on the same dataset, but by putting more weight on the wrongly classified samples, such that we obtain a simple yet accurate model at the end of the training phase.

Gradient boosting [49]: as opposed to AdaBoost, gradient boosting does not adapt the weights of the training samples for each new tree, but uses gradient descent to adjust the new model's parameters to minimize a loss function.

XGBoost [50]: XGBoost stands for eXtreme Gradient Boosting. XGBoost is an optimized implementation of gradient boosting, using system optimization and algorithm enhancements. This algorithm is a very widely (and successfully) used in Kaggle competitions.

Stacking [51]: stacking stands for Stacked Generalization. With this protocol, we aim at learning a model to learn the models. More precisely, we combine the inferences of different so-called base ML models and consider these base outputs as new input data for a meta-model, while still aiming at estimating the ground-truth value from the original dataset. This meta-model thus learns how to best combine inferences to improve accuracy.

Evaluation Strategies

We briefly discuss how we can evaluate ML models by explaining two commonly used evaluation strategies:

Test-set method: this is the simplest assessment we can use. We divide our dataset into two parts, with random sampling used to decide to which subset each sample belongs to:

- The training set, on which we train/build the model,
- The test set, on which we let the model carry out inferences.

We finally analyze the quality of these estimations to get an idea about the performance of the model. Very often, we use between 70%-80% of the data for training, and the remaining part for testing.

***k*-fold cross-validation:** when using this protocol, we randomly subdivide our dataset into k parts. For each of these parts, we first learn a model on the data *not* in the considered part, i.e. on the $k-1$ other folds. We then evaluate the trained model on the data reserved for testing. With this approach, we ensure that each sample is used once for model assessment. Typically, we choose $k = 5$ or $k = 10$ depending on the size of the dataset.

Evaluation Metrics

We introduce the most widely used metrics for the evaluation of ML models. The choice of the evaluation metric mostly depends on the type of the inference task (regression/classification) and on the application itself.

We denote by y_i the ground truth for the ith sample and $\hat{y}_i$ the estimated value for the same sample.

For regression, the following metrics are very common:

Mean absolute error (MAE): $\frac{1}{N} \cdot \sum_{i=1}^{N} |\hat{y}_i - y_i|$. The absolute value avoids that errors with different signs cancel each other.

Root mean squared error (RMSE): $\sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} |\hat{y}_i - y_i|^2}$. RMSE penalizes much more the large errors than MAE.

Mean relative error (MRE): $\frac{1}{N} \cdot \sum_{i=1}^{N} \frac{|\hat{y}_i - y_i|}{y_i}$.

Pearson linear correlation coefficient (PLCC): $\frac{\sum_{i=1}^{N}(\hat{y}_i - \overline{\hat{y}}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(\hat{y}_i - \overline{\hat{y}})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \overline{y})^2}}$ where $\overline{y} = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i$. A PLCC value close to 1 indicates that the real and estimated values are strongly positively correlated, a negative value shows a negative correlation, and a PLCC close to 0 indicates that there is no linear correlation.

For classification, a very basic measure is the **overall accuracy**, which is defined as the ratio of correctly classified samples over the number of total data points. However, this metric is not useful when dealing with a highly imbalanced dataset, and even misleading in that case.

To retrieve more detailed information about the performance of an ML model, we introduce the following notions for binary classification. Those can be easily generalized to multi-class classification.

True positive/negative (TP/TN): a sample that has been correctly classified as positive/negative respectively.

False positive/negative (FP/FN): a sample that has been wrongly classified as positive/negative respectively.

A handy tool to summarize the performance of a binary classifier is the **confusion matrix**. It includes the four previously mentioned metrics, or the corresponding rates defined as follows:

True positive rate (TPR): $\frac{TP}{TP+FN}$. It's also called **sensitivity** or **recall**.

True negative rate (TNR): $\frac{TN}{TN+FP}$. It's also called **specificity**.

False positive rate (FPR): $\frac{FP}{FP+TN}$.

False negative rate (FNR): $\frac{FN}{FN+TP}$.

Another very important metric is the **precision**, which is defined as $\frac{TP}{TP+FP}$. The precision tells us what proportion of inferred positive samples are actually positive. This score should be very high if false positives are a problem (such as for spam detection).

We also introduce the **F1 score**, which takes into account both the precision and recall. This can be useful if there should be a balance between those two. It is defined as $2 \cdot \frac{precision \cdot recall}{precision + recall}$.

As a last notion, we present **receiver-operating-characteristic (ROC) curves**. They help understand the performance of binary-classification models at all classification

thresholds and show the different trade-offs between the false-positive rates (FPRs) and true-positive rates (TPRs). ROC curves are only defined for two classes; therefore, when there are more than two classes, we plot one such curve for each class versus all the other ones ("one versus all"). To summarize a ROC curve in a single number, we use the **area under the ROC curve (AUC)**. The machine-learning community often uses the AUC statistic for model comparison, which is simple and informative, and provides more reliable comparisons when dealing with imbalanced data. To explain the AUC, let us consider a 2-class classification model f that first estimates a probability of a sample x belonging to the class $+$, denoted by $f(x)$, then uses this probability to make a decision. If x^+ is a random sample of the class $+$ and x^- another random sample of the class $-$, the AUC is the probability that $f(x^+) > f(x^-)$. The higher the AUC, the better the discrimination of the model. The AUC also reveals how good the recall and precision of a model for a specific target class are: higher AUC values reflect higher TPR values and lower FPR values.

Feature Selection

In this subsection, we present feature selection, which is used to reduce the number of features. Reducing the number of features when building models has several benefits: it helps avoid overfitting and thus potentially increases the model's performance, it makes the model easier to interpret, and it might reduce the computation time.

We can group feature-selection approaches into three categories:

1. **Filter techniques.** With these, we calculate a relevance score for each feature based on its relationship with the inference target and remove the ones with the lowest scores from our feature set. To compute these scores, we can use simple statistical methods such as the information gain, a chi-squared test, etc. Filter techniques are completely independent from the considered supervised algorithm.

2. **Embedded techniques.** As the term already suggests, they are embedded within the learning algorithms themselves. More precisely, the most relevant features are selected during the training process. A good example is the decision tree, whose training is based on ranking the different features based on their impurity reduction to select the best one for the node splits. The final feature importance is calculated as the sum over the nodes in which that feature is tested of the decreases in node impurities weighted by the probability of reaching the corresponding nodes [38]. For a forest, the resulting variable importances are the averages over all trees.

3. **Wrapper techniques.** With these methods, we evaluate subsets of features with the model we want to use for training and use the one which maximizes its quality. One example is Recursive Feature Elimination (RFE) [52]: the model is trained n times, n being a hyperparameter fixed by the user, and, at each iteration, we remove the weakest feature.

2.4.2 Unsupervised Learning

As opposed to supervised learning, we do not provide the models with labeled data in unsupervised learning. Instead, the aim is to discover patterns in the data, without any guidance. In particular, the role of this domain of ML is threefold:

- to search for interesting groups of variables and/or samples (i.e. searching for clusters)
- to detect anomalies
- to reduce the dimensionality of data, i.e. the transformation of high-dimensional data into data fitting into a low-dimensional space, in such a way that the important properties of the original data are preserved

For clustering, we can mention two important algorithms:

***k*-Means [53]:** this is the simplest clustering algorithm. The number of clusters (k) needs to be determined in advance. The algorithm starts by randomly selecting k centers among the available data points. Each sample is then assigned to the closest center (in terms of Euclidean distance, typically). Next, a new center gets calculated for each cluster by computing the component-wise mean of all the samples belonging to the same cluster. The samples get now reassigned to the clusters with the new centers. These steps are repeated until the assignments do not change anymore.

DBSCAN [54]: DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. Unlike for k-means, we do not need to specify the number of clusters in advance; this number is automatically detected by the algorithm using the notion of density (an area is dense if it contains many data points). In a nutshell, the algorithm proceeds by selecting a point arbitrarily and deciding whether it forms a cluster with nearby points: if the distance to these other points is below a variable threshold *and* if there are sufficiently many such points (this is another parameter of the algorithm), they are considered part of the same cluster. The algorithm stops as soon as all data points are either assigned to a cluster or decided to be noise.

When it comes to anomaly detection, we briefly introduce two algorithms:

Isolation forest [55]: this algorithm behaves similarly to randomized trees, but at each node, both the feature and the cut point are chosen randomly. The number of nodes a sample needs to traverse to reach a leaf is the normality measure, such that the fewer nodes a sample has to visit, the more abnormal it is. This is intuitive, as outliers have unusual characteristics, and are thus rapidly distinguishable from the normal samples.

Local outlier factor [56]: an algorithm that relies on the concept of local density for detecting outliers. The local density of a sample is estimated from the distance to its k nearest neighbors. The algorithm compares the local densities of the given sample and its k nearest neighbors and if a sample has a much lower density than they do, it is considered as an outlier.

Finally, we present the most well-known algorithm for dimensionality reduction:

PCA [57]: PCA stands for Principal Component Analysis. In a nutshell, PCA allows us to linearly transform a large set of variables into a smaller one that contains only uncorrelated variables, without losing any important properties. To achieve this, the algorithm maps the data into the user-specified number of dimensions, while trying to preserve its variance as much as possible. There exist two equivalent formulations to tackle this problem: the maximum-variance formulation to search for directions that maximize the variance of the new data, and the minimum-error formulation to minimize the reconstruction error of the new data.

2.4.3 Active Learning

Active learning (AL) is a variant of supervised learning where not all labels are known: it is up to the active-learning algorithm to decide which labels are worth acquiring. That choice can be carried out using different methods, like inference uncertainty, for instance. This paradigm is especially useful when acquiring labeled data is expensive. A good example is the domain of cybersecurity: determining whether an Android app is malicious requires manual intervention of highly-skilled security researchers.

The main distinction in active-learning algorithms is the frequency at which new data is obtained:

- In **pool-based active learning**, the whole dataset is available from the beginning, with some samples labeled and others not; the algorithm must decide which labels should be retrieved.
- In **stream-based active learning**, new data points are continuously flowing into the system; the algorithm must decide, typically very quickly, whether a newly arrived sample should be labeled.

As active learning is part of supervised learning, we still need a small set of labeled data points for training. After this initial training, the model is free to choose the samples among the unlabeled ones it wants to learn from.

2.4.4 Reinforcement Learning

Reinforcement learning (RL) is a very different learning paradigm, due to the central importance of interactions with an environment [58]. Some interactions are beneficial

Figure 2.10: Interactions in RL.

and should be encouraged: they are generally associated with a positive reward; other behaviors should be avoided and they thus generate lower rewards, potentially negative ones (i.e. penalties).

Reinforcement-learning algorithms provide a way to build an agent which interacts with the environment, with the goal of maximizing the total reward the agent gathers each time it performs an action.

A couple of notions, besides the terms of agents and rewards, are important in reinforcement learning:

Environment: it comprises everything outside the agent, typically the outside world; for instance, the road for a self-driving car, or the computer network in which an intelligent router is deployed.

State: the RL-driven agent measures the state of the environment before taking any decision: a car that is on the edge of a cliff should not speed up (the state indicates that the car is near the ledge), an intelligent router should strive to avoid sending more packets into an already congested link (the state describes the congestion level of the outside links).

Action: any action that the agent performs has the potential to influence the next state of the environment (forwarding more packets into a congested link will only worsen the blockage, for instance). The action to take is decided with the goal of maximizing the overall reward over a large period of time: the agent is not forced to take decisions that yield immediate rewards, but is instead encouraged to move to states that are more likely to generate excellent rewards in the long term.

The interactions between the different parts are depicted in Figure 2.10.

A particular case in RL is the domain of bandit algorithms. They consider a simplified version of the problem where the environment has only a single state, and the actions

taken by the agent do not influence in any way this state. Bandits are typically evaluated using a metric called the regret, defined as the difference between the maximum reward the agent can get from the environment on average and the actual reward it obtained. The literature presents many examples of extremely efficient bandit algorithms (as measured by the regret) for several scenarios, like multi-armed bandits [59], linear bandits [60, 61], or linear contextual bandits [60, 62].

CHAPTER 3

Video-QoE Monitoring and Analysis

Notice

Parts (text as well as figures, tables, and algorithms) of this chapter have already been published in the following publications:

[J3] *ViCrypt to the Rescue: Real-time, Machine Learning-Driven Video QoE Monitoring for Encrypted Streaming Traffic*
S. Wassermann, M. Seufert, P. Casas, G. Li, L. Kuang
IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2007-2023, 2020
DOI: 10.1109/TNSM.2020.3036497

[W2] *I See What You See: Real Time Prediction of Video Quality from Encrypted Streaming Traffic*
S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li
Workshop on QoE-based Analysis and Management of Data Communication Networks, October 2019
DOI: 10.1145/3349611.3355549

[C3] *On the Analysis of YouTube QoE in Cellular Networks through In-Smartphone Measurements*
S. Wassermann, P. Casas, M.Seufert, F. Wamser
Wireless and Mobile Networking Conference, September 2019*
DOI: 10.23919/WMNC.2019.8881828

[J1] *Machine Learning Models for YouTube QoE and User Engagement Prediction in Smartphones*
S. Wassermann, N. Wehner, P. Casas
ACM SIGMETRICS Performance Evaluation Review, vol. 46, no. 3, pp. 155-158, 2018
DOI: 10.1145/3308897.3308962

[C2] *Beauty is in the Eye of the Smartphone Holder – A Data Driven Analysis of YouTube Mobile QoE*
N. Wehner, S. Wassermann, P. Casas, M. Seufert, F. Wamser
Conference on Network and Service Management, November 2018*

YoMoApp and the data- and feature-acquisition modules of ViCrypt have been developed by the researchers of the university of Würzburg. The paragraphs describing those modules have been mostly written by this team, even though I edited them for clarity (Sections 3.2.1, 3.3.1, 3.3.2 *Dataset Acquisition*, and 3.3.3). My work focused on the data-analytics and machine-learning parts and I wrote the corresponding sections.

Video streaming is the dominant application of today's Internet. According to Sandvine's Global Internet Phenomena Report [63], video streaming made up 61% of the global Internet application download traffic in 2019. The same company found that video streaming was very popular during the stay-at-home period in the context of the COVID-19 pandemic in 2020 [64]: YouTube accounted for 15% of the global traffic and Netflix 11%. Netflix, YouTube, and Amazon even decided to reduce the default playback resolution to avoid a service breakdown during that time. Futuresource further predicts that video viewing will represent more than 80% in 2022 [65].

As a consequence, ISPs strive to deliver a high video-streaming QoE, in order to satisfy their clients and avoid customer churn. The intensifying competition among the different ISPs is forcing them to integrate QoE into the core of their network-management systems, from network monitoring and reporting to traffic engineering. The goal of network operators is not only to operate their networks efficiently, but also to avoid severe degradations of the subjective experience. QoE measurements represent today a major source of information to monitor, analyze and subsequently manage operational networks.

Traditionally, network-traffic monitoring has relied on Deep-Packet-Inspection (DPI) techniques to analyze the performance of Internet services and applications. DPI approaches were particularly handy for video-streaming analysis, as investigating the payload of the packets containing video information could be used to understand the status of the video-player buffer [66, 67] and gave insight into the video-playback performance. Unfortunately, ISPs can no longer use those methods as they have lost access to this kind of information due to the deployment of TLS encryption.

Today, we are facing an additional challenge, the one of the ever growing number of mobile devices. Most of the end users nowadays access the Internet via cellular networks to consume contents, from Web browsing and video streaming to a plethora of novel services offered through apps. Indeed, according to Ericsson, there were more than 6

billion mobile broadband subscriptions globally in November 2020 [68]. The increase in the volume and heterogeneity of contents consumed in cellular networks and on mobile devices forces cellular-network ISPs to not only focus on the quality of their fixed-line networks, but also on their cellular networks which come with their own issues, such as user mobility.

In this chapter, we first dive into the related work done on QoE of adaptive video streaming. We then present two systems for QoE monitoring:

1. YoMoApp, an Android application collecting session-based KQIs after a user watched a YouTube video. We conceived an AI solution based on YoMoApp data for the inference of session-based QoE metrics exclusively from network-level features.

2. ViCrypt, a platform retrieving QoE metrics in real time while a user is watching a YouTube video on a desktop machine. An ML module allows the real-time inference of those metrics from network statistics.

For both frameworks, we detail their functioning, the evaluations we conducted with them, and the corresponding results we obtained. The last section concludes this work.

3.1 Related Work

In the past, video streaming mostly suffered from waiting times, namely stalling, caused by re-buffering events [69, 70, 71], and also from initial delay, i.e. the time until the start of the playback [72]. In the last years, these degradations have been partially mitigated by adapting the video bitrate to the network conditions, using HTTP Adaptive Video Streaming (HAS) or Adaptive Bitrate (ABR) streaming technologies. To operate HAS video streaming, the video content must be available in multiple bitrates, i.e. quality levels, and split into small segments or chunks, each containing a few seconds of playtime. The client-side adaptation logic requests the next chunk of the video in an appropriate bitrate, such that the initial delay is minimized, stalling is avoided, and the quality level is maximized to best utilize the available bandwidth. The decisions of the adaptation logic are typically based on the current bandwidth and/or buffer status [73, 74], but might take into account other aspects, such as client characteristics or fairness among competing clients [75]. The HAS streaming technology is adopted by a wide range of applications and video content providers, such as YouTube, Netflix, and Amazon, and has been standardized as MPEG-DASH in ISO/IEC 23009-1 [76].

Changing the video bitrate also means modifying the visual quality of the streamed video, e.g., in terms of resolution, frame rate, or compression, which introduces an additional impact on QoE. An interesting finding on the impact of HAS on QoE [77, 78, 79, 80] is that, rather than quality changes, the most relevant metric to monitor is the fraction of time during which the video is played out at a high visual quality: the higher this fraction, the better the QoE. As a consequence, ISPs are highly interested in solutions

to estimate video-resolution levels, which can serve to detect events when the played-out quality level drops as soon as they happen, to take appropriate countermeasures. For example, thinking towards a more proactive network-management paradigm, ISPs would like to additionally estimate and predict the video bitrate to adjust the network configuration in time. This could include appropriately shaping the allocated bandwidth or selecting suitable routes for the streaming traffic, which would avoid further QoE degradation.

In [81], authors presented YoMo, an in-device, application and DPI-based tool for YouTube-QoE monitoring, capturing video-player activity and buffering conditions to infer re-buffering events. In [67, 82, 66], they extended YoMo and its overall concept to monitor YouTube QoE in cellular and fixed-line networks at scale, using DPI approaches. Others [83, 84, 85] adopted similar in-application measurements for YouTube-QoE monitoring, relying on application-side tools to directly collect KQIs such as playback delay, re-buffering events, video resolution, or quality switches. Application-side monitoring provides accurate measurements for QoE assessment, as these can be directly observed, without the need of additional estimation or mapping approaches. As mobile Internet access gets more and more important, there are also numerous tools which are developed with the purpose of monitoring QoE in cellular networks and on smartphones, such as Netalyzr [86] and Mobilyzer [87]. Smartphone-app QoE can be monitored with QoE Doctor [88], an active-measurement tool analyzing both network- and application-layer features. Other tools for measuring YouTube QoE in smartphones are introduced in [89] and [90].

As already mentioned earlier, the trend towards end-to-end encryption makes it much harder for network operators to have a good picture of the traffic of their customers. It is no longer possible to rely on DPI to analyze the video data contained in each packet and reconstruct the streaming process and the video buffer [67], or to intercept and analyze segment requests. The encrypted stream of packets offers only very basic information about the streaming process, such as packet sizes and inter-arrival times, which brings ML-based approaches more and more to the center of the academic and industrial attention. For example, in [2, 91, 92], authors apply different ML approaches to estimate QoE-relevant metrics for YouTube by extracting features from the stream of encrypted packets, generally using simple features such as packet sizes or throughput. Similarly, authors in [93] follow a machine-learning-based analysis to infer QoE metrics for YouTube streaming over cellular networks. Other papers propose to reconstruct the evolution of the buffered video playtime [66], by analyzing the encrypted stream of packets through heuristics and statistical modeling approaches [94]. In particular, [95] considers the data in the form of TLS transactions to perform the considered QoE inferences. In [96], authors present Prometheus, a system using machine-learning models to infer the QoE of smartphone apps through passive network measurements.

3.2 Session-Based Mobile Video-QoE Monitoring and Analysis

In this section, we present YoMoApp, an Android application allowing users to watch YouTube videos in an embedded YouTube video player while collecting a rich set of measurements such as traffic statistics and video-quality metrics like stalling events and video-quality switches. We also detail the analyses we performed with this tool. The contributions to the networking community are as follows:

- **Presentation of a rich dataset:** our dataset is very rich in terms of user diversity and spans a long measurement period. It thus provides a unique insight into mobile-end-user YouTube QoE.
- **Mobile QoE modeling:** we built ML models to infer crucial application-layer QoE metrics exclusively relying on network-layer measurements collected through YoMoApp.
- **Comparison of QoE-assessment models:** YoMoApp gives the user the possibility to rate the quality of the session, which allows us to compare different QoE models to real user feedback.

In Section 3.2.1, we describe the application, and in Section 3.2.2, we detail the dataset collected with YoMoApp. Next, we use the YoMoApp dataset to carry out a temporal analysis of YouTube performance in Section 3.2.3. In Section 3.2.4, we discuss and compare different QoE models. As a last step, we infer key video-QoE metrics with machine-learning models in Section 3.2.5.

3.2.1 YoMoApp – the YouTube Monitoring App

Note about adoption of text from co-authors: The text in Section 3.2.1 was mainly contributed by co-authors to the joint paper [C3], even though I edited it for clarity.

YoMoApp [23] is an Android application which is freely available on the Google Play Store and can be run both on smartphones and tablets. It provides a distributed monitoring platform for YouTube QoE, collecting user feedback in a crowd-sourced manner, and passively measuring a large set of QoE-relevant KPIs at the player and network side. Metrics such as stallings, initial playback delay, and video resolution are retrieved at play time. YoMoApp additionally collects QoE feedback provided by the user, once a video is fully watched or aborted. A simple questionnaire with multiple questions allows the user to rate the QoE of the video session according to a standard 5-level ACR MOS scale [27], ranging from bad (MOS = 1) to excellent (MOS = 5). Questions include the user's feedback on the quality of the video, the quality of the streaming, the user's opinion on the video content, as well as the service acceptability (yes/no). The QoE-feedback questionnaire is presented to the user only if she wishes to provide such feedback, which is specified at the YoMoApp starting time.

All YoMoApp measurements are periodically uploaded to a remote server, building a comprehensive database of YouTube performance- and QoE-related measurements.

YoMoApp replicates the original YouTube mobile Website in functionality, design, and video-playback performance. An Android `WebView` is embedded to display the YouTube Website, using an HTML5 `video` element relying on adaptive-streaming technology for the video playback. The monitoring is done at runtime via JavaScript, which queries the HTML5 `video` element.

YoMoApp relies on JavaScript event listeners to monitor changes of the player state ("playing", "paused", "buffering", and "ended", for instance), and the resolution of the video element. The app monitors the current playback time and the buffered playtime every second. Additionally, it retrieves metadata, as for example the YouTube video ID, title, and duration of the watched video. The gathered data is then sent to and processed by the application. As the usage of JavaScript is prone to inconsistencies and errors, like missing/incorrect values or non-equidistant data queries, the data is post-processed locally by YoMoApp.

Besides playback events, YoMoApp measures both network and context features. Moreover, it collects device features such as size of the screen, orientation, playback audio volume, size of the player, and playing mode ("full screen", for example). Lastly, the application gathers network-traffic statistics such as per-second uploaded/downloaded bytes, as well as information such as GPS-based location, cellular operator, ID of the cell, Radio Access Technology (RAT), or strength of the signal. The current version of YoMoApp extracts the session state on the user's device, which could also be done in the network, but this inference task is difficult in its own right and we leave it as future work.

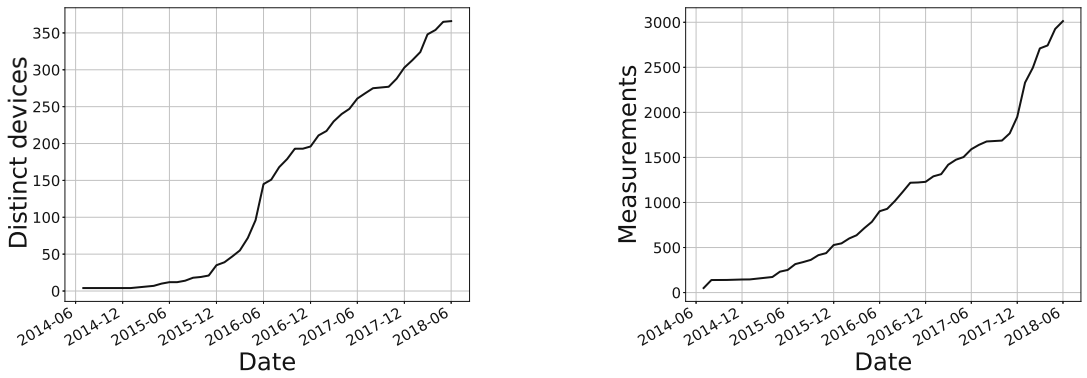
3.2.2 YoMoApp Dataset

In this subsection, we analyze the dataset retrieved with the help of YoMoApp. The dataset spans five years, namely the period from 2014 to 2018. It includes more than 3,000 YouTube video sessions collected worldwide over 70 distinct cellular ISPs and from more than 360 different users. These users were scattered throughout multiple regions of the world, with most of them in Europe (Germany, France, Austria, and the UK), but also with some residing in the US. The heat map in Figure 3.1 depicts the geographical distribution of the recorded video sessions: they are distributed all over 58 different countries. About 38% of the measurements were carried out in Germany, 17% in Greece, 9% in India, and 5% in France. Measurements gathered in other countries represent a share equal or less than 3% each.

Figure 3.2 reports the accumulated number of YoMoApp measurements over time and the cumulative number of unique devices, respectively. A surge of new users is clearly observed from 2016 on, which comes as a consequence of a stronger advertisement of YoMoApp performed by the researchers leading this project at that time and an increased dissemination through different research communities and conferences. The numbers of

Figure 3.1: Distribution of the YoMoApp measurements worldwide.

streamed sessions, new users, and collected measurements have more than doubled since January 2017. Interestingly, there were more than 900 new sessions during the first half of 2018, largely exceeding the number of measurements monitored in 2017.



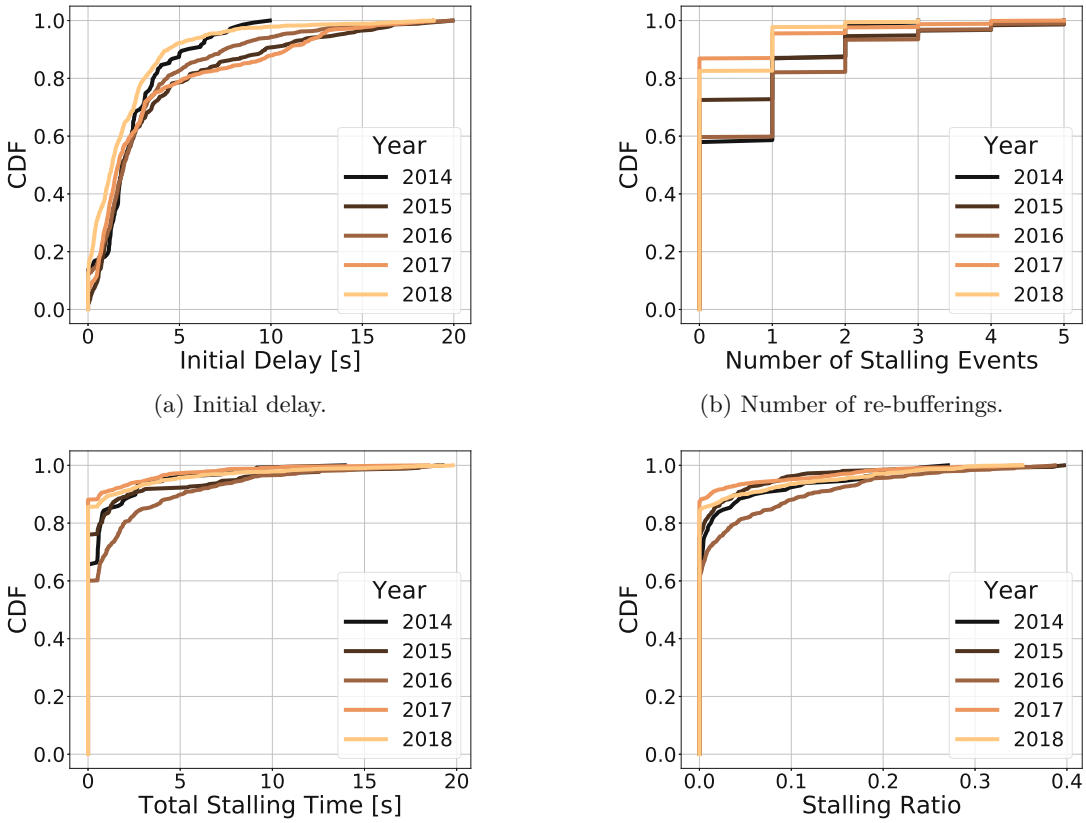
(a) Cumulative number of devices.

(b) Cumulative number of video sessions.

Figure 3.2: Cumulative number of measurements and different devices over time.

3.2.3 Temporal YouTube QoE Analysis

We now study the evolution of the most important YouTube KQIs over time. In particular, we study the evolution of initial delays, number of stallings, stalling times, and re-buffering ratios. Figure 3.3 depicts the empirical distribution of these metrics per year. A first observation is that there is a clear enhancement of all QoE-related metrics along time, 2018 being the year with the best performance in terms of initial playback delays and re-buffering events.



(a) Initial delay.

(b) Number of re-bufferings.

(c) Total stalling time.

(d) Stalling ratio.

Figure 3.3: Temporal evolution of the performance of YouTube mobile streaming in terms of QoE-relevant KPIs.

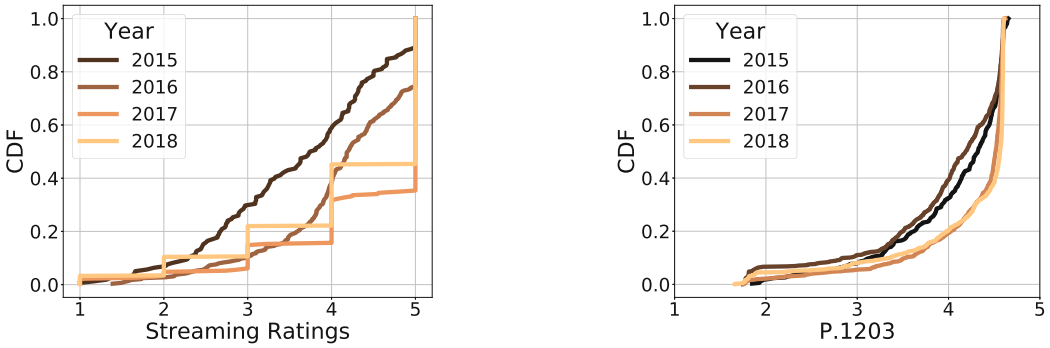
About 90% of the sessions in 2018 have an associated initial delay below 5 seconds, and we observe a similar fraction for videos streamed and displayed without stalling. On the other hand, the initial delay for video sessions in 2016 was below 5 seconds for about 80% of the videos, and only 60% of the videos were played without re-buffering events. Furthermore, more than 15% of the videos in 2016 suffered from a re-buffering ratio higher than 10%, whereas this fraction falls to about 5% in 2017/2018.

We also show in Figure 3.4 that such an improvement is reflected by the QoE feedback reported by the end users. In particular, this figure depicts the distribution of the subjective MOS scores as provided by the users (Figure 3.4a) and an estimation of the MOS scores, obtained by the ITU-T P.1203 model for adaptive video streaming [28] (Figure 3.4b). As we can see, the users' feedback is accurately captured by the P.1203 model. We focus on the QoE of the users regarding their opinion on the video-streaming performance. As we can observe in Figure 3.4a, the users indicated MOS scores through a continuous scale before 2017, and using a discrete scale from 2017 on to make the

interface more intuitive.

As already stated earlier in this subsection, the quality of YouTube streaming increased significantly in the last two considered years. Indeed, in 2017 and 2018, we have more than 80% of the videos rated with MOS scores equal or above 4, while this fraction had a value between 40% to 60% in previous years.

Finding 1: YouTube QoE on mobile devices has improved over the past years.



(a) Subjective ratings.

(b) P.1203 scores.

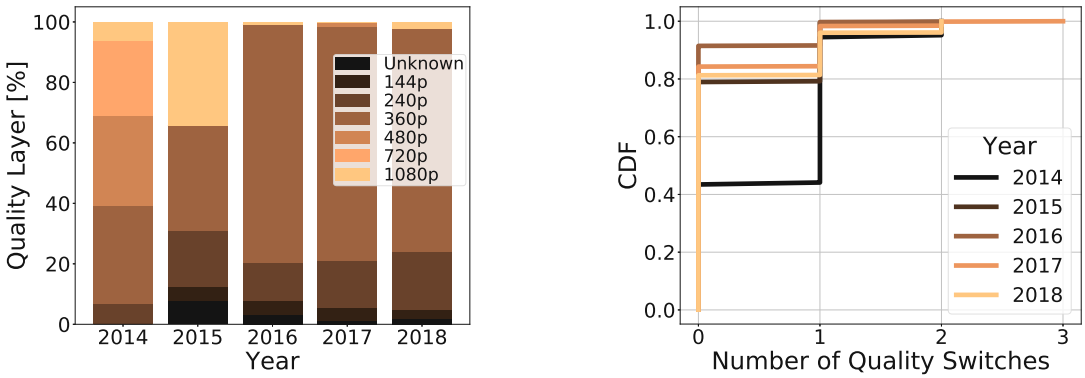
Figure 3.4: Distribution of MOS scores per session.

The distribution of requested video qualities by the YoMoApp video player reveals that the played-out video qualities varied much more back in 2014 and 2015 in contrast to the period from 2016 to 2018, with a higher prevalence of higher quality levels as compared to today. Figure 3.5a shows the distribution of the main quality levels per session grouped per year. The YouTube streaming service has been evolving over time, not only for the fixed-line network scenario, but also in mobile networks. When YouTube started playing on mobile devices, the adaptive-streaming policy was less conservative and higher quality levels would be requested in the adaptive-streaming mode. From 2016 onward, the dominant video quality changed to 360p, which is a more conservative quality level, imposing less bandwidth requirements. There are also videos with lower resolutions like 144p or 240p, but almost no HD content was streamed within the last three years with YoMoApp. This is perfectly aligned to previous findings on YouTube QoE in smartphones [97], where the authors observed that lower vertical resolutions result in the same subjective experience as higher resolutions when dealing with smartphones, due to the small screen sizes. Thus, streaming HD content makes less sense and is less efficient in mobile devices than lower qualities.

Thanks to this conservative behavior, it is also not surprising that the number of quality switches observed within the last three analyzed years is significantly lower compared to

2014 and 2015. Figure 3.5b displays the distribution of the number of quality switches per session. In more than 80% of the sessions, no quality switch could be observed for the period of 2016 to 2018, meaning that the initial quality selected by YouTube was matching the underlying network performance. In contrast, in 2014, only 43% of the sessions showed no quality switch, around 53% observed one quality switch, and the remaining sessions resulted in two or more quality switches.

Finding 2: YouTube mobile video distribution is more efficient today than in the past.



(a) Video-quality-level distribution.

(b) Distribution of the number of switches.

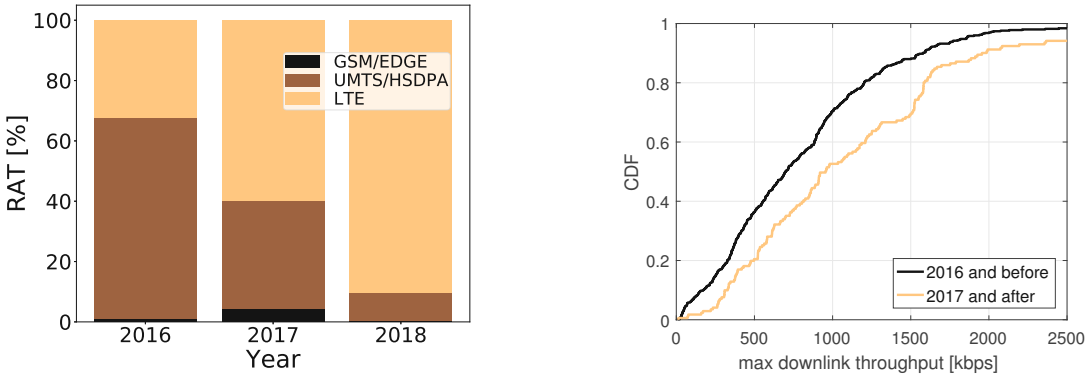
Figure 3.5: Video-quality levels and quality switches.

As a last step, let's take a look at the mobile technologies, in particular RAT, and user engagement. For the RAT, we differentiate between 2G (GSM, EDGE), 3G (UMTS, HSDPA), and 4G (LTE). RAT information started being collected only from 2016 on. In 2016, 3G was the dominant RAT, with a prevalence of about 66% of all sessions with cellular access. In 2017, the balance shifted and 4G became the dominant RAT with a share of 59%. This dominance increased even more in 2018, where sessions with LTE make up 90% of all streaming sessions with cellular access. Figure 3.6a depicts the distribution of the main RAT per session per year. As a consequence of the RAT evolution, we observe better network performance over time. For example, Figure 3.6b shows the distribution of the maximum download throughput achieved by YoMoApp video sessions before and after December 2016. The average maximum download throughput increased from about 2 Mbps to more than 10 Mbps, and the median has also increased from about 600 kbps to 1 Mbps.

User engagement is defined as the fraction of the total video length a user watched before the video was aborted or the video ended (100% user engagement). Figure 3.7

reports the user-engagement distribution per year. YoMoApp started to measure the user engagement only in 2015, we therefore have no results for 2014. Results show how user engagement has systematically increased over time, and significantly in 2018: more than 60% of the videos were watched completely and only 20% of the users aborted the video at 20% or less of the video playback. This indicates that YoMoApp is more and more being used as a standard video player. The increased user engagement can also be explained by the improvement of the network performance in terms of higher downlink throughputs. We note that video-duration averages and distributions are similar across the different years, ruling out potential bias on user engagement.

Finding 3: Mobile network technology and performance have also improved, potentially resulting in increased user engagement.



(a) Radio Access Technology (RAT).

(b) Maximum downlink throughput.

Figure 3.6: Radio access technology and video download throughput.

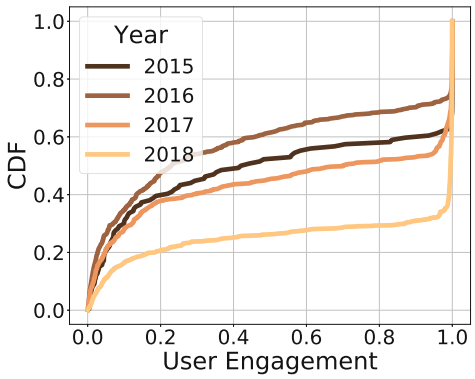


Figure 3.7: Evolution of user engagement.

3.2.4 QoE Modeling and Assessment

In this subsection, we focus on the problem of YouTube-QoE modeling and assessment, calibrating different YouTube QoE models available in the literature as well as standardized models – in particular, the ITU-T P.1203 model.

As already observed in Section 3.2.3, the subjective MOS ratings obtained from the users are accurately modeled with the P.1203 model.

Besides the application of the P.1203 model, we additionally investigate simpler QoE models available in the literature [98, 99]. These models are of exponential nature, under the form

$$f(\alpha, \beta, \gamma, \delta, L, N) = \alpha \cdot e^{-(\beta \cdot L + \gamma) \cdot N} + \delta$$

where α, β, γ, and δ are parameters that need to be calibrated using the specific dataset being analyzed, and L and N correspond to the average stalling length and number of stallings respectively. We take a simple manual calibration approach to set $\alpha = 4$ and $\delta = 1$ to be within the MOS range. In particular, we want the values of f to be close to 1 (i.e. bad quality) when we are faced with numerous and/or long stallings and close to 5 in case the playback runs smoothly. We apply non-linear-least-squares regression to set the other parameters.

Another family of models we look at are referred to as simple additive QoE models, which are expressed as a linear combination of individual univariate models:

$$Q(x_1, \ldots, x_n) = \sum_{i=1}^{n} w_i \cdot Q_i(x_i)$$

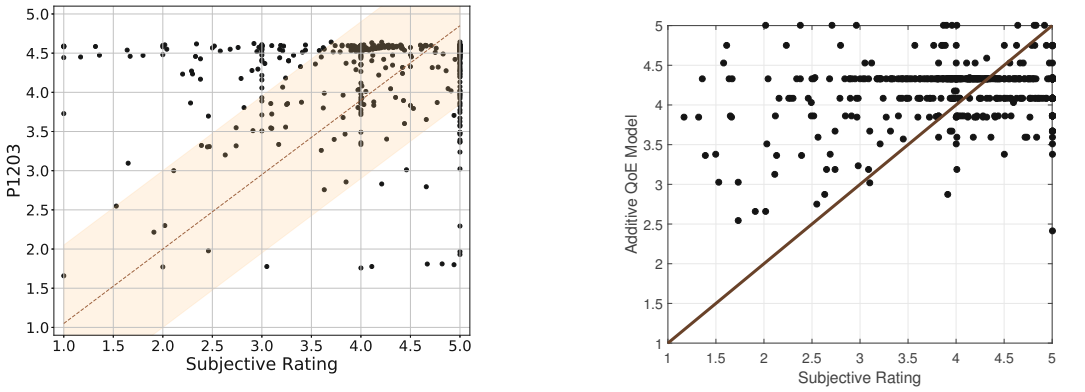
where weights w_i are greater or equal to 0, $\sum_{i=1}^{n} w_i = 1$, and $Q_i(x_i)$ is of exponential nature just as f, but takes into account only a single QoE metric x_i [99]. We rely again on non-linear-least-squares regression to determine the values of the parameters to tune. Our evaluation revealed that the additive QoE model expressed as

$$\begin{aligned} 0.49 \cdot \left(4 \cdot e^{-0.14 \cdot \#stallings} + 1\right) + 0.17 \cdot \left(4 \cdot e^{-47.7 \cdot initialDelay} + 1\right) \\ + 0.34 \cdot \left(4 \cdot e^{-0.44 \cdot \#qualitySwitches} + 1\right) \end{aligned}$$

is the one which fits best the data.

Figure 3.8 depicts two scatter plots reporting the modeled MOS scores vs. the actual user subjective feedback. Figure 3.8a reports the results for the P.1203 model, whereas Figure 3.8b considers the best model out of the two tested ones [98, 99], which corresponds to the additive QoE model. In both cases, we observe that both QoE models tend to overestimate the actual QoE ratings reported by the users. This suggests that users might be actually more annoyed than what one could perceive by directly using these QoE models in practice.

Lastly, Figure 3.9 depicts the linear correlations observed between both the subjective ratings and the P.1203 estimations and application-layer metrics such as stalling, initial



(a) P.1203 vs. subjective ratings.

(b) Additive QoE model vs. user feedback.

Figure 3.8: Modeled MOS scores vs. actual user feedback.

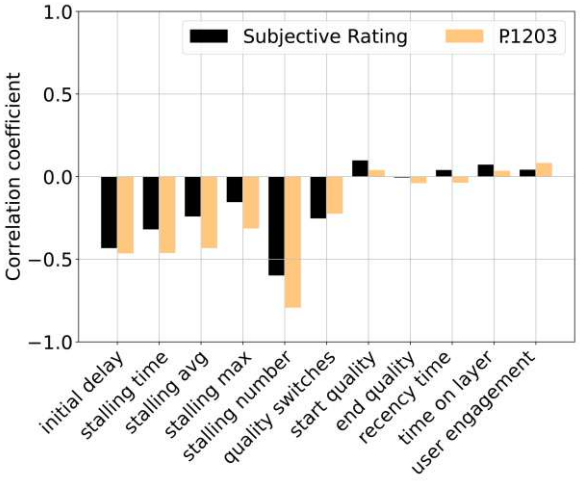


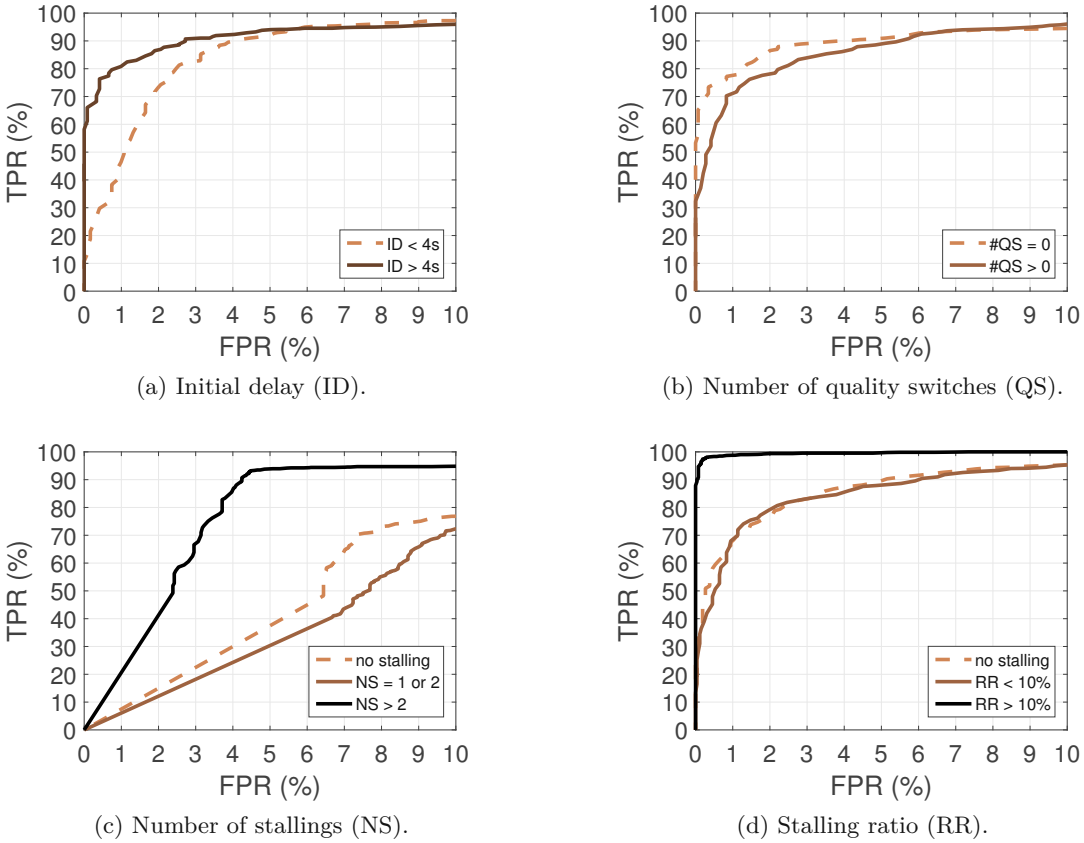
Figure 3.9: Linear correlations – subjective ratings and P.1203.

delay, quality switches, and up to user engagement. While the correlations tend to be rather low, there is a clear negative impact of stalling duration, initial delay, and number of stallings on both QoE values (feedback and P.1203), as observed in past studies.

3.2.5 QoE Inference Through Machine Learning

As a last study, we focus on the inference of QoE-relevant metrics which are normally measured directly by YoMoApp, but assuming that we only have access to the mobile devices' general network-level measurements, available through the Android APIs. The rationale is that we would like to monitor YouTube mobile KQIs such as initial delay, stalling, and quality switches, but without using an application like YoMoApp. These estimators could be applied in a more generic mobile-device-based monitoring

system, where users would not be forced to run an app with an embedded player such as YoMoApp to measure relevant KPIs, and where such KPIs could be actually estimated for any user watching YouTube videos on her device, independently of the YouTube player being used. We do not take into account the hardware used for streaming (e.g., the video-decoding performance of the device) and focus on the network layer. Indeed, a previous study [100] showed that the hardware of the smartphone is not a strong predictor of the user QoE.



(a) Initial delay (ID).

(b) Number of quality switches (QS).

(c) Number of stallings (NS).

(d) Stalling ratio (RR).

Figure 3.10: QoE-metrics estimation performance. ROC (one-versus-all) curves highlight high recall for the considered classes against the other ones.

We tackle the inference of four key video-QoE metrics. We build estimators using ML models, treating each of the four problems as a classification task, where targets are discretized. The reason why we choose to address our inference tasks as classification problems is that ISPs are more interested in the overall performance of their services

than in exact numbers, i.e. want to mainly know whether a given KPI is above/below a certain threshold or not. The four classification targets are as follows:

1. **Initial delays:** whether initial delays (ID) are above or below a pre-defined QoE-relevant threshold. Based on previous work on initial delay tolerance [101], we set this value to four seconds; 77% of the sessions in our dataset have an ID below four seconds.
2. **Video-quality switch:** whether a video-quality switch (QS) has occurred during the session or not; more than 80% of the sessions did not experience any quality switches.
3. **Stalling event:** the number of stalling events (NS), grouped into three classes:
   - **Zero stalling:** 85% of the sessions.
   - **Mild stalling:** one or two stalling events; 10% of the sessions.
   - **Severe stalling:** more than two stallings; 5% of the sessions.
4. **Re-buffering rate:** the stalling frequency or re-buffering rate (RR), considering again three classes:
   - **Stalling-free:** 85% of the sessions.
   - **Mild stalling:** stalling events occurred and lasted for less than 10% of the total duration of the video session; 10% of the sessions.
   - **Severe stalling:** stallings occurred for more than 10% of the whole session; 5% of the sessions.

For each metric, we train and evaluate a random-forest model with 10 trees through 10-fold cross-validation. We rely on simple bootstrapping techniques to balance classes for learning purposes. For the inference, we use the network-layer features captured by YoMoApp, which can actually be measured by simply accessing the Android developer APIs. The full feature set encompasses 275 features (see complete list in Appendix A), including information about the received signal strength, the number of handovers, the number of network switches, and multiple statistics about the incoming and outgoing traffic, aggregated at different time windows of 1, 5, 10, 30, and 60 seconds. The traffic is measured on three different levels: the total traffic transmitted/received by the device, the traffic captured over the mobile network, and the traffic sent/received by the application itself.

Figure 3.10 reports the obtained results for the inference of the four KQIs in terms of ROC curves. The curves are interesting as they show the trade-offs between TPRs and FPRs. Our results are fairly accurate for the four estimation targets, achieving good classification rates for most of the classes. For example, the initial-delay discrimination as well as the quality-switching detection can be done with a false positive rate below 5%

for more than 90% of the sessions. Results are even better when inferring the re-buffering ratio, with an almost perfect performance for detecting bad-quality sessions with a high stalling ratio. Inferring the number of stalling events is clearly more challenging than that of the other three targets.

We use feature-selection techniques to identify the most relevant features for each estimation target. In particular, we rely on an embedded approach within a 10-tree random forest. We find that about 30 features out of the 275 are needed to obtain accuracies highly similar to the ones achieved with the full feature set using the same type of model. In particular, features derived from the received and transmitted traffic are very important. Indeed, statistics about the traffic generated by the application itself and by the device are among the top features, including metrics describing the variation of the throughput across multiple time-window lengths. Metrics related to changes in the signal strength are also among the selected features. Interestingly, the number of handovers does not seem to play an important role for the QoE-metrics inference in our case.

3.3 Stream-based Video-Quality-Metric Inference

With YoMoApp, we focused solely on session-based QoE estimations, we now go one step further and infer video-quality metrics in real time. In this section, we present ViCrypt, an ML-driven monitoring solution able to infer the most important KQIs for HAS, namely stalling, initial delay, video resolution, and average video bitrate. The contributions to the QoE community are as follows:

- **Fine-grained, real-time operation:** ViCrypt estimates the most important KQIs, i.e., initial delay, stalling, and visual quality, as well as the video bitrate in real time during the streaming of a video session with a fine-grained temporal resolution of just one second. To the best of our knowledge, this is the smallest granularity proposed so far in the literature, enabling quick anomaly detection and troubleshooting approaches, as well as proactive traffic management.
- **Stream-based feature computation in constant memory without requiring chunk detection:** different from all previously presented proposals, ViCrypt continuously extracts features from the encrypted stream of packets in a stream-like manner, using a bounded and low memory footprint. This enables the execution of ViCrypt on top of memory-constrained hardware, such as home routers, which are nowadays the preferred devices for conducting end-customer monitoring by major vendors. Indeed, ViCrypt targets the monitoring of video-streaming QoE from devices installed near the end user, which do not necessarily belong to the ISP – but to the vendor – and where traffic load enables real-time monitoring with limited hardware. ViCrypt features are based on packet-level statistics and their computation does not require chunk-detection mechanisms, removing the extra overheads and errors introduced by such a detection step.

- **Fine-grained estimation:** ViCrypt tackles substantially more precise estimation tasks than previous work [2, 102]. In fact, ViCrypt estimates the six most common different video-resolution classes – 144p, 240p, 360p, 480p, 720p, and 1080p – instead of discriminating between low and high resolution. In addition, it provides a continuous estimation of the average video bitrate by relying on regression techniques.
- **Extensive ML-model benchmarking:** also unlike previous work, we devote a significant part of this study to benchmark different ML algorithms and evaluate their performance using different sets of inputs, carefully engineered by feature-selection approaches to limit the quantity of required input data for proper execution.
- **Empirical validation over a heterogeneous YouTube dataset:** last but not least, we show that ViCrypt performs accurately under a very heterogeneous set of scenarios. For this, we empirically tested the system over a large dataset of more than 15,000 streaming sessions of different YouTube videos. The dataset covers different access technologies (WiFi and LTE), transport protocols (QUIC and TCP), bandwidth configurations, players, and devices (browser player in laptops and native YouTube application in smartphones). Measurements were collected at four different ISPs in four different EU countries. This is an additional advantage over the state of the art, where proposals are generally validated using fewer or less representative scenarios.

In Section 3.3.1, we give an overview of ViCrypt and compare it to the state of the art. In Section 3.3.2, we present the dataset we collected in the context of this study. In Section 3.3.3, we detail the features used by ViCrypt for the QoE estimations. In Section 3.3.4, we introduce the machine-learning models we use in our benchmarking. In Section 3.3.5, we evaluate the different ML models and report the results. As a next step, in Section 3.3.6, we investigate which features are the most important ones for the considered QoE metrics. Finally, in Section 3.3.7, we run ViCrypt on two different machines and observe its running times.

3.3.1 Introducing ViCrypt

Note about adoption of text from co-authors: The text in Section 3.3.1 was mainly contributed by co-authors to the joint paper [J3], even though I edited it for clarity.

In this subsection, we present ViCrypt [24], an ML-driven monitoring solution capable of estimating and continuously tracking the most relevant KQIs for HAS, in real time, and using input features derived from raw network-traffic measurements. As it has been shown in previous work [102, 93, 3, 2, 103, 104, 105], it is possible to extract features from the (encrypted) stream of packets which are strongly correlated with different QoE-relevant metrics. The targeted KQIs include stallings, initial playback delay, video resolution, and average video bitrate. To do so, ViCrypt analyzes ongoing streaming

sessions using fine-grained time slots of 1-second length, computing multiple lightweight, statistical features from the video traffic in a stream-based fashion. At the design phase of ViCrypt, we tested similar temporal resolutions – up to 5 seconds – without resulting in significant changes in performance, yet loosing monitoring resolution. Besides per-time-slot features, ViCrypt additionally computes features for two different temporal aggregations of past slots:

1. A short-term memory capturing the last t slots, also called *trend* windows.
2. A long-term memory, aggregating all time slots since the start of the video session, also called *session* windows.

At the end of each 1-second time slot, all these described features are fed into ML models, which estimate the video resolution, average bitrate, and stalling of this slot. Capturing statistics at different time scales is very beneficial, as some quality degradations build up in a few time slots or even instantly, while others are the result of longer-term phenomena.

ViCrypt provides fine-grained estimations, either by building classification models with many quality classes – such as for video-resolution analysis – or by building regression models – as for video-bitrate analysis, for instance. Such a combined fine-grained temporal scope and estimation resolution allows ViCrypt to provide better and sharper insights into video HAS QoE than other solutions.

The two most similar approaches to ViCrypt are Requet [3] and the system presented in [2]. ViCrypt improves on both in multiple aspects:

1. While both approaches claim to be real-time, there is no evaluation of the computational costs required during the feature-extraction procedures, questioning their claims; in addition, for some of the targets, Requet has a temporal resolution based on chunk lengths (typically several seconds of a video), and [2] operates at a 10-second time scale, both significantly higher than for ViCrypt. This impacts their usability in practice for critical real-time monitoring applications, such as troubleshooting.
2. While Requet also provides the same fine-grained classes for estimation of video resolution as ViCrypt, estimations in [2] are coarser-grained, with just a few classes. Moreover, video bitrate, which is especially relevant for ISPs, is not inferred by their systems.
3. While ViCrypt operates directly on the stream of packets at the network and transport layers, Requet requires chunk-detection mechanisms to extract chunk-based features, which is error-prone and introduces additional delays and complexities.

Other relevant differences between our study and the previous studies presented in [3] and [2] are reported in Table 3.1; it offers a comprehensive overview on the properties of

		ViCrypt	Requet	INFOCOM'18[2]
KQI estimation target	Initial delay (# classes)	✓ (continuous)	✗	✓ (binary)
	Stalling (# classes)	✓ (binary detection / continuous estimation)	✓ (binary detection)	✓ (binary detection)
	Resolution (# classes)	✓ (6 levels, 144p - 1080p)	✓ (6 levels, 144p - 1080p)	✓(binary, ≷ 480p)
	Bitrate (# classes)	✓ (continuous)	✗	✗
Input features	Chunk detection required	✗	✓	✗
	# features	208	127	226
	Feature selection	✓ (down to 20 features)	✗	✗
Network monitoring	Real-time	✓	✓	✓
	Temporal resolution	1 second	5 seconds or every chunk	5 - 10 seconds
	Feature computational efficiency	✓	not tested	not tested
Training / evaluation data	Streaming service	YouTube	YouTube	YouTube
	# video sessions	15,000+	600	11,000
	Access network	WiFi & cellular	WiFi	not mentioned
	# ISPs | geo-location	4 ISPs | 4 EU countries	2 ISPs | US / India	1 ISP
	Devices	laptop & smartphone (native app)	laptop	laptop
	Time span	9 months in 2018 / 2019	6 months in 2018	4 months in 2017

Table 3.1: ViCrypt vs. Requet and INFOCOM'18 [2]. Overview of the properties of the proposed solutions in terms of type and detail of the inferred KQIs, input features, monitoring capabilities, and datasets used for training and testing purposes.

the proposed solutions in terms of type and detail of the inferred KQIs, input features, monitoring capabilities, and datasets used for training and testing purposes.

3.3.2 YouTube Dataset

In this subsection, we first detail how we retrieved our dataset before diving into its analysis.

Dataset Acquisition

Note about adoption of text from co-authors: The text in this section (Data Acquisition) was mainly contributed by co-authors to the joint paper [J3], even though I edited it for clarity.

Over a period of several months from June 2018 to February 2019, we streamed and recorded more than 15,000 YouTube video sessions, resulting in a total of more than 4,600,000 1-second time slots. For reference, Requet [3] collected measurements for only

580 video sessions, resulting in a dataset almost 26 times smaller. As we describe next, our dataset is not only large in terms of number of video sessions, but also very diverse.

For the video-streaming and data-collection tasks, we used a monitoring tool similar to [106]. It relies on the Selenium browser-automation library to automatically start a Chrome browser and randomly select a YouTube video to stream. Chrome was configured such that all HTTP requests were logged and QUIC traffic was enabled. A JavaScript monitoring script [23, 107] was injected into the Webpage to record the current timestamp every 250 ms, as well as the current video playtime, buffered playtime, video resolution, and player state.

We streamed the video sessions with very diverse network setups to reach a highly generalizable model. Video sessions were collected at home (30% of the samples) and over corporate WiFi networks (50%), as well as over LTE mobile networks (20%). For 60% of the sessions, a firewall was enabled, which blocked all QUIC traffic, such that the videos were streamed via TCP. The maximum bandwidth was roughly 20 Mbps. Additionally, some streaming sessions faced bandwidth limitations, which were applied to limit both up- and downlink traffic. The bandwidth limitations were either constant on a level of 300 kbps, 1 Mbps, 3 Mbps, or 5 Mbps, or they fluctuated between these levels every one to five minutes. The number of video sessions per bandwidth configuration is mostly balanced among the different categories, with a slightly higher number of sessions streamed without bandwidth limitations. We collected video sessions from four different geographic locations – France, Austria, Germany, and Italy – and from four different ISPs.

Network traffic was collected for each video-streaming session, logging basic per-packet information (timestamp, source IP address, source port, destination IP address, destination port, size), as well as DNS-lookup responses to obtain a mapping between IP addresses and domain names. In each network-traffic trace, we identified YouTube video flows based on domain names (`googlevideo.com`), and extracted features only for these video flows, ignoring all non-YouTube traffic. Finally, we also included in our measurements the recently published YouTube open dataset [108], which includes measurements from the native, mobile Android YouTube application. While the share of app measurements is limited compared to desktop devices (less than 10%), it contributes to the heterogeneity of the learning data.

Finally, we did not consider the challenging issue of video-traffic detection and filtering in this work, besides the aforementioned DNS-based identification approach. This is indeed a complex issue, especially when considering multiple users sharing the same IP address – e.g., NAT. Indeed, running multiple video sessions in parallel without disentanglement would lead to wrong network statistics, as we would not be able to identify which packets belong to which videos, for instance. The specific video-traffic identification and disentangling of concurrent video-streaming sessions is out of our scope.

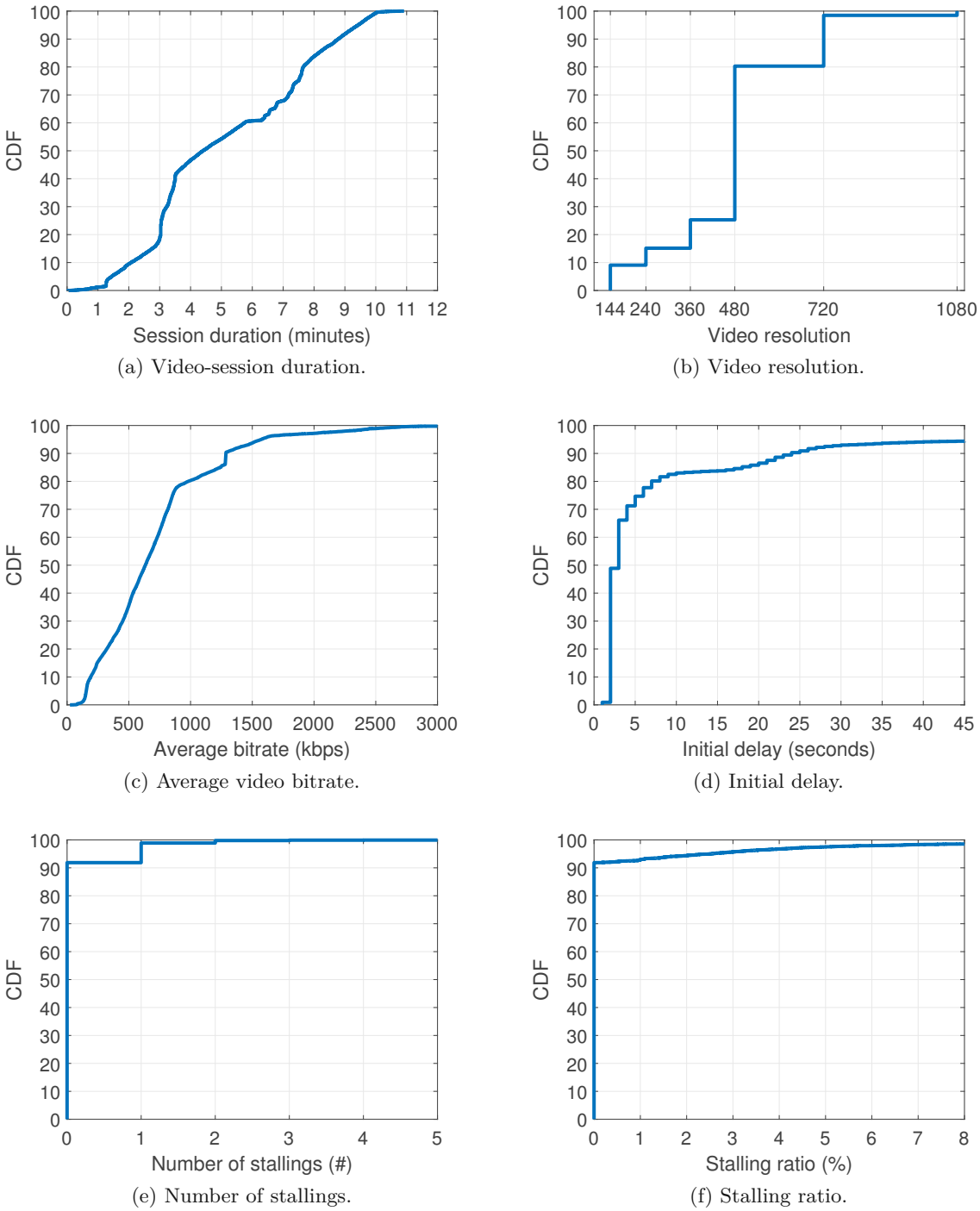
Dataset Analysis

Next, we provide some insight into the collected dataset. Figure 3.11 shows the characteristics of the dataset as cumulative distribution functions (CDF), regarding the duration of the video sessions, the video resolution, the average bitrate, the initial delay, and stalling. Figure 3.11a shows that the recorded video sessions have durations between a couple of seconds and 11 minutes. The average length of a video session is approximately 5 minutes.

YouTube typically indicates the video resolution as the number of vertical pixels, for which standard quality classes exist. The video-resolution classes contained in the dataset can be easily observed in Figure 3.11b, namely 144p, 240p, 360p, 480p, 720p, and 1080p. In some videos, the video resolution did not match exactly one of these classes, and was therefore rounded to the nearest class. The distribution shows that the adaptation logic of YouTube decided to stream videos mostly in 480p resolution, but also very low resolutions occur (9% 144p, 6% 240p, 10% 360p). At the other end, HD resolution is rare (18% 720p, 1% 1080p). We did not observe any resolutions above 1080p during the measurements; thus, although supported by YouTube, we kept resolution classes to the 6 observed in the dataset – the same 6 resolution levels were considered by Requet [3].

Figure 3.11c depicts the distribution of the average bitrate. Here, the average bitrate was obtained via the YouTube API, and represents the average bitrate of the full video when streamed with a given quality level (itag). Thus, this estimation target is not the momentary bitrate of the current slot, but rather the average bitrate of the quality level that was downloaded in the current slot. The average video bitrate spreads from approximately 20 kbps to about 4,600 kbps. Nearly all of the slots have an average bitrate less than 3,000 kbps. The CDF increases steeply, almost uniformly, until roughly 900 kbps, which corresponds to 78% of the slots. Then, it increases slower only showing a steep increase around 1,280 kbps, which is the average bitrate for 4% of the slots, and thus seems to be a certain target bitrate for encoding.

Figure 3.11d shows the distribution of the initial delays. Nearly 50% of the sessions have an initial delay of at most two seconds, and 75% of the sessions have a delay below five seconds. Figures 3.11e and 3.11f depict the distribution of two stalling metrics, namely the number of stallings per video session and the stalling ratio, i.e. the fraction of time spent in stalling mode with respect to the full session duration. Stalling events are rare: more than 90% of the videos do not stall at all, and when they do, the large majority stalls only once. The stalling ratio suggests that most of the stalling events are short with respect to the session duration: more than half of the observed stalling ratios are at most 3%.



(a) Video-session duration.
(b) Video resolution.
(c) Average video bitrate.
(d) Initial delay.
(e) Number of stallings.
(f) Stalling ratio.

Figure 3.11: Characterization of the YouTube dataset, composed of more than 15,000 video-streaming sessions. © IEEE 2020

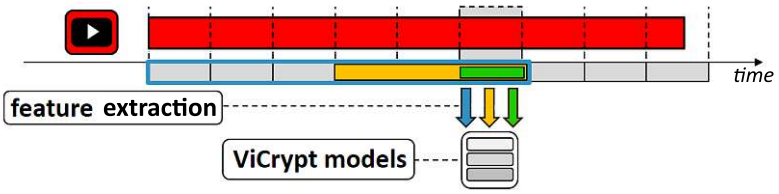


Figure 3.12: ViCrypt session overview (video in red). We extract the session features from the blue time slots, the trend features from the yellow ones, and the features about the current state from the green one.

3.3.3 ViCrypt Feature Extraction

Note about adoption of text from co-authors: The text in Section 3.3.3 was mainly contributed by co-authors to the joint paper [J3], even though I edited it for clarity.

We now detail how the feature extraction works within our framework. Figure 3.12 presents a general overview of the functioning of ViCrypt. ViCrypt operates in a time-slotted, sequential manner, producing estimations for the selected KQIs at the end of each elapsed time slot. Features are extracted and continuously updated for each new packet on the stream of encrypted video traffic; at the end of each new time slot, different ML models infer the corresponding KQI from the extracted features.

ViCrypt embeds temporal notions in the construction of features, using information not only from the current time slot, but also from past slots. For memory and computational efficiency, the past streaming information must be compressed and structured. This is why ViCrypt keeps track of two additional macro windows, referred to as *trend* or short-term memory window, and *session* or long-term memory window. The trend window comprises the last t time slots in a sliding-window fashion, i.e., it contains all traffic of the current time slot and the $t-1$ most recent slots. In the context of this study, ViCrypt uses a trend size of $t = 3$, and thus the features of the trend window of each time slot are computed from the traffic of the current slot and the previous two time slots. The value of t is set using grid search on the evaluated dataset, but as opposed to the time-slot length, the trend window length has a non-negligible impact on the model performance. The proposed value provides the best results in terms of inference performance for the analyzed data.

The second macro window is the session window, which includes all traffic of the session so far observed, and its features are therefore extracted from the traffic in all previous slots including the current time slot. All features of each current slot, trend window, and session window are computed in an online, stream-based fashion, without the need to store the previous traffic or detailed information about traffic packets observed in the past. This technique significantly reduces the memory consumption of the feature-extraction process – from linear to constant –, enabling a lightweight monitoring solution.

Next, we dig deeper into the feature-extraction process, elaborating on the different computation steps.

Table 3.2 briefly summarizes the features computed per time slot. All features are derived from three basic, packet-level metrics, namely packet count, packet size, and inter-arrival time (IAT). Different aggregations are done on these metrics, based on individual time slots and aggregation windows – trend and session ones. Features are computed for uplink, downlink, and total traffic. The rationale behind the computed features is rather straightforward: adaptive video-streaming protocols employ closed-loop algorithms to achieve synchronization and adaptability between server and video player, thus traffic patterns on both uplink and downlink direction might reveal different behaviors at the player side.

up/down/total	**Volume**	**Throughput**	**Distribution**	**Protocol shares**
packet size	✓	✓	✓	✓
packet count	✓			✓
packet IAT		✓	✓	

Table 3.2: ViCrypt features. All features are derived from three basic packet-level metrics, namely packet count, packet size, and inter-arrival time (IAT), aggregated at time-slot-based and window-based resolutions.

Firstly, we compute simple count-based features from the traffic observed in the time slot. These consist of the number of total, uplink, and downlink packets, and the number of transferred bytes (total, uplink, downlink). We also count the number and byte volume of TCP and UDP packets, and compute the upload ratio, download ratio, TCP ratio, and UDP ratio from these counters, for both number of packets and number of bytes. Next, we extract time-based features. These include the time from the start of the slot until the first packet, the time after the last packet until the slot ends, and the burst duration, i.e., the time between the first packet and the last packet of the time slot. All features are again computed for the total traffic, as well as for uplink and downlink traffic. The average throughput of the slot (traffic volume divided by slot length) and the burst throughput (traffic volume divided by burst duration) can be subsequently derived for total, uplink, and downlink traffic. A covariance-based procedure is used to obtain a linear regression for the cumulative traffic over time, in an online fashion [109]. Algorithm 3.1 stores the origin of the regression (start time of the time slot $\texttt{slot}_{\texttt{start}}$), and keeps updates of the number of packets $\texttt{num}_{\texttt{P}}$ and the cumulative packet size $\texttt{cum}_{\texttt{size}}$, to compute the regression for the cumulative traffic. In addition, it stores and updates the current means of the abscissa (time, $\texttt{mean}_{\texttt{T}}$) and ordinate (cumulative packet size, $\texttt{mean}_{\texttt{S}}$), the number of packets $\texttt{n}$, as well as the temporal variance $\texttt{var}_{\texttt{T}}$ and covariance $\texttt{cov}_{\texttt{TS}}$. These are the only permanently stored variables, updated whenever a new packet of size $\texttt{s}$ arrives at time $\texttt{t}$.

The updates to these statistics only use three additional temporal variables: $\texttt{diff}_{\texttt{slot}}$, $\texttt{d}_{\texttt{t}}$, and $\texttt{d}_{\texttt{s}}$. At the end of the slot, we compute the final slope (`slope`) and intercept (`intercept`) values from the regression curve. We perform two regressions for uplink

and downlink traffic, and the slope and intercept of these regression curves are added as features.

Finally, we extract multiple features derived from the empirical distribution of the traffic. We use an algorithm based on [110], which can compute the first four statistical moments of any distribution in an online fashion, i.e., the mean, the variance, the skewness, and the kurtosis. We extend this algorithm to additionally output the standard deviation, the coefficient of variation, as well as the minimal and the maximal values.

Here again, only few statistics are stored in memory and updated, namely the number of packets `n`, the mean value `mean`, the second, third, and fourth centered statistical moments ($\texttt{sdm}_2$, $\texttt{sdm}_3$, $\texttt{sdm}_4$), which are defined as:

$$\texttt{sdm}_i = \sum_{t=1}^{n} (x_t - \texttt{mean})^i \qquad \forall i \in \{2, 3, 4\}\,,$$

as well as the minimal (`min`) and the maximal (`max`) values. The update of these statistics occurs whenever a new value `x` is observed for the corresponding statistic. The approach is explained in Algorithm 3.2.

The computed updates allow to directly obtain the mean (`mean`), minimal (`min`), and maximal (`max`) values. Moreover, the variance (`var`), standard deviation (`std`), coefficient of variation (`cvar`), skewness (`skew`), and kurtosis (`kurt`) of the distributions can be computed as stated in Algorithm 3.3. These distribution-based features are computed for the packet size and the IAT, for both uplink and downlink traffic.

This results in a total of 69 basic features for the traffic in a time slot, plus the same 69 basic features for each of the two additionally considered macro windows, namely the trend and the session windows. Together with the sequence number of the current time slot, which is also included as a feature, this sums up to a total of 208 features, characterizing each slot of 1-second length (see complete list in Appendix B. To keep track of the trend windows of size t, we have to maintain and update not only the current trend window, but additionally $t-1$ future trend windows. These future trend windows are the windows which will become trend windows in $1, \ldots, t-1$ windows, but already have to consider and aggregate the traffic of the current time slot. In contrast, only a single session window is needed, as it only needs to accumulate the full traffic of the complete session. Thus, in total, $t+2$ windows with 69 features each have to be maintained and updated at all times, i.e., current time slot, trend window, session window, and $t-1$ future trend windows.

3.3.4 ML-Model Benchmarking

Solutions so far proposed in the state of the art such as [102, 93, 3, 2, 103, 104, 105] rely mostly on random-forest models as the underlying ML approach. We provide further insights into the performance of different types of models, benchmarking 11 different ML models within ViCrypt. 9 out of these 11 models can be trained for both classification

Algorithm 3.1 Online regression computation.

procedure COMPUTEREGRESSION(s, t)
  $n \leftarrow n+1$
  $cum_{size} \leftarrow cum_{size} + s$
  $diff_{slot} \leftarrow t - slot_{start}$
  $d_t \leftarrow diff_{slot} - mean_T$
  $d_s \leftarrow cum_{size} - mean_S$
  $var_T \leftarrow var_T + \frac{\frac{n-1}{n} \cdot d_t^2 - var_T}{n}$
  $cov_{TS} \leftarrow cov_{TS} + \frac{\frac{n-1}{n} \cdot d_t \cdot d_s - cov_{TS}}{n}$
  $mean_T \leftarrow mean_T + \frac{d_t}{n}$
  $mean_S \leftarrow mean_S + \frac{d_s}{n}$
  $slope \leftarrow \frac{cov_{TS}}{var_T}$
  $intercept \leftarrow mean_S - slope \cdot mean_T$
end procedure

Algorithm 3.2 Online update of distribution metrics, used for computation of distribution features. The algorithm is executed when new values are observed.

procedure UPDATEDISTRIBUTIONS(x)
  $n \leftarrow n+1$
  $d_x \leftarrow x - mean$
  $d_n \leftarrow \frac{d_x}{n}$
  $mean \leftarrow mean + d_n$
  $sdm_4 \leftarrow sdm_4 + [d_x\, d_n\, (n-1)\, d_n^2\, (n^2 - 3n + 3)] + (6\, d_n^2\, sdm_2) - (4\, d_n\, sdm_3)$
  $sdm_3 \leftarrow sdm_3 + [d_x\, d_n\, (n-1)\, d_n\, (n-2)] - (3\, d_n\, sdm_2)$
  $sdm_2 \leftarrow sdm_2 + [d_x\, d_n\, (n-1)]$
  if $x < min$ **then**
    $min \leftarrow x$
  end if
  if $x > max$ **then**
    $max \leftarrow x$
  end if
end procedure

Algorithm 3.3 Computation of distribution features.

1: **procedure** COMPUTEDISTRIBUTIONFEATURES
2: $\texttt{var} \leftarrow \frac{\texttt{sdm}_2}{\texttt{n-1}}$
3: $\texttt{std} \leftarrow \sqrt{\texttt{var}}$
4: $\texttt{cvar} \leftarrow \frac{\texttt{std}}{\texttt{mean}}$
5: $\texttt{skew} \leftarrow \sqrt{\frac{\texttt{n}}{\texttt{sdm}_2^3}} \cdot \texttt{sdm}_3$
6: $\texttt{kurt} \leftarrow \texttt{n} \cdot \frac{\texttt{sdm}_4}{\texttt{sdm}_2^2} - 3$
7: **end procedure**

and regression tasks, while the other two are designed for anomaly detection, hence our selection.

Most of the selected models also rely on decision trees, not only because of their proven high accuracy and low computational cost, but also due to a series of embedded properties, such as model explainability, robustness to input noise, and embedded feature selection. We consider both averaging and boosting ensembles based on trees, which brings robustness, increased accuracy, and improved generalization of the trained model. To increase training speed, reduce model complexity, and therefore reduce the chances of overfitting, we favor small-sized ensembles, using 10 to 50 models.

Model hyperparameters are calibrated through standard grid-search optimization for the stalling-inference task; we reuse the same hyperparameters for the two remaining tasks. Finally, all evaluations throughout this study are done through 5-fold cross-validation. When it comes to the classification tasks, we apply stratified cross-validation, i.e., we ensure that the five folds preserve the percentage of samples for each class. The list of benchmarked models includes:

- Decision tree (DT)
- Random forests with 10 trees (RF10)
- Adaboost using 50 trees (ADA)
- Ensemble with 10 extremely randomized trees (ERT10)
- Bagging with 10 trees (Bagging)
- Naive Bayes (Bayes)
- k-nearest neighbors with $k = 5$ (kNN)
- Neural network with three hidden layers (NN): the first layer containing 200 neurons, the second 100 neurons, and the last 50 neurons. For each hidden layer, we use the sigmoid function as the activation function, and for the output layer, we use the softmax function.

- Support-vector machine (SVM)

As stalling can be considered as an anomaly of the streaming process, we additionally evaluate two anomaly-detection algorithms for stalling *detection*:

- Isolation forests with 10 trees (ISO10)
- Local outlier factor with the number of neighbors set to 20 (LOF)

When considering DT, RF10, and ERT10 models, and to counterbalance the impact of imbalanced classes, we assign weights to each sample i of $class_i$ based on the occurrence frequency of its class:

$$W_i = \frac{\#samples}{\#classes \cdot (\#samples\ in\ class_i)}$$

This implies that the estimation errors for samples of rare or under-represented classes are significantly penalized, which improves the estimation accuracy in theory for these classes. We further exploit the fact that RF10, ERT10, Bagging, and kNN models can be parallelized for speed improvement, and run them in a parallelized fashion. For NN, we use TensorFlow on GPU, while we use the scikit-learn library for the remaining models. The benchmark of the models is executed on a high-end desktop computer, equipped with two Intel Xeon Silver 4116 processors including 12 physical cores each (a total of 48 virtual cores thanks to Intel HyperThreading), 128 GB of RAM, and a NVIDIA GeForce RTX 2080 Ti graphics card (with 11 GB of VRAM).

3.3.5 ViCrypt in Action – Performance Evaluation

We now present the performance-evaluation results of ViCrypt for all the described KQIs. We take the full set of 208 features as input, i.e., we do not explicitly consider feature-selection approaches. We devote Section 3.3.6 to feature selection. For each of the tested ML algorithms, we report performance metrics, as well as the total running time for training and inference, i.e., the time needed to compute the 5-fold cross-validation on the whole dataset. This helps better understand the practical trade-offs when it comes to real-time analysis.

Stalling Estimation

As stalling is the most severe QoE degradation, our first goal is to estimate whether the video is stalling or not. More precisely, for each 1-second time slot, ViCrypt infers whether the video is being played or stalling; this is therefore a binary classification problem. We consider only time slots which contain network traffic, and end up with almost 1,283,000 samples.

The binary stalling-estimation results obtained for each of the time slots can be further combined to obtain stalling metrics at a video-session level, such as the initial playback

	Accuracy (%)	Recall (%)	Precision (%)	5-CV time (min)
DT	96	64	68	57
RF10	97	55	88	3
ADA	95	29	61	154
ERT10	97	54	88	1
Bagging	97	65	87	63
Bayes	50	86	9	1
kNN	96	48	71	10
NN	94	0	0	600
SVM	84	62	21	36
ISO10	86	13	8	4
LOF	86	11	6	46

Table 3.3: Benchmarking of ML models for the stalling detection. To compute the overall accuracy, the recall, and the precision, we consider the stalling class as the positive one.

delay, the number of stalling events, and the stalling ratio – ratio of total stalling time to total playback time. The initial delay is given by the number of slots inferred as stalling at the start of the session. The number of consecutive slots with stalling is added to the total stalling time in seconds. Thus, simply counting slots with their estimated stalling status allows to obtain the initial delay, the number of stalling events, the total stalling time, and the stalling ratio of the whole streaming session. The granularity of the initial-delay and stalling-time estimation is limited by the time-slot length, one second. Nevertheless, such a fine-grained resolution is sufficient for most monitoring use cases.

We therefore present evaluation results for stalling at two different temporal granularities:

1. **Per slot:** binary stalling classification (stalling/no stalling).
2. **Per session:** continuous estimation of initial delay, number of stalling events, and stalling ratio.

Table 3.3 summarizes the overall accuracy, recall, and precision for the stalling class per model, as well as the overall cross-validation times. The recall indicates the percentage of time slots of a given class for which ViCrypt correctly inferred the playback state. In contrast, the precision for a given state expresses the proportion of the class estimations which are correct. Results show that stalling detection is a challenging task, especially due to the high imbalance of the data (cf. Figure 3.11e). Indeed, let us take the NN model as example: the trained model was not able to identify a single stalling slot, still achieving a high accuracy of 94.3%, due to the imbalance. This highlights that the overall accuracy can be misleading with highly imbalanced data, and that it is

particularly important to look in detail at recall and precision results. Here, we observe that the tree-based models achieve a high precision of around 90% for most of them, but only a recall of around 55%. Bagging is an exception, with a recall of 65%.

To dig deeper into these results, Figure 3.13 presents the confusion matrices for the different models. We left out the confusion matrix of the neural network, due to its poor performance as discussed above, and the ones for RF10 and LOF, as the matrices are nearly identical to those of ERT10 and ISO10, respectively. The confusion matrices underline once again that stalling detection is a rather difficult task. Surprisingly, Bayes is the algorithm yielding the highest stalling-class accuracy (i.e., recall), achieving 86%. However, its very poor precision turns the approach inapplicable. The outlier-detection algorithms (ISO10 and LOF) also perform poorly for this estimation task, which might indicate that the feature values do not deviate much between the stalling and no-stalling classes.

ERT10 seems to be a good model choice for stalling detection: it runs very quickly and reaches a decent recall-precision combination. Only Bagging achieves an overall better performance, especially a higher recall, but at the cost of a much higher cross-validation processing time of roughly one hour versus one minute for ERT10. Of course, as training is usually done offline, this would in principle not be a limitation for Bagging. However, if one would consider adaptive learning approaches, or applying the model in scenarios with strong video-traffic variations, low training times become paramount.

As a final analysis for this model, we additionally try imbalance-correcting techniques, such as SMOTE [111] and random oversampling of the majority class. When randomly oversampling the stalling class, the results are highly similar to the ones without re-balancing. With SMOTE, ERT10 achieves a much higher recall – almost 74% – but the precision drops by 17 percentage points to 71%. Moreover, the running time gets multiplied by 10, which is a significant drawback when speed is important.

We now explore the inference performance of ViCrypt for stalling metrics at the session level. More precisely, we estimate the initial playback delay of the videos, the number of stalling events, and the stalling ratio. Figure 3.14 shows the distribution of estimation errors for the three considered targets. Regarding initial delay, ViCrypt perfectly infers the real playback delay for about 40% of the video sessions, and achieves an error of at most 2 seconds for 70% of the sessions. The number of stallings is perfectly estimated for about 50% of the sessions, and an error of at most 2 stallings is realized for about 75% of the sessions. The stalling ratio is perfectly estimated for about 60% of the sessions, and the error is below 3% for more than 85% of the sessions. Errors related to stalling estimation usually correspond to over-estimations, which is always preferred from the point of view of ISPs, for the sake of safety margins and over-provisioning. Also, stalling ratio has been the preferred metric in the state of the art when it comes to session-based stalling estimation [93, 102], and our results are in line with or even better than the state of the art [93, 102], even when dealing with such a strong imbalance in the data.

Finally, for visualization purposes, Figure 3.15 shows the real-time stalling estimation

produced by ViCrypt for an exemplary YouTube video-streaming session, using ERT10 as underlying model. ViCrypt can track in real time the overall stalling pattern of the video session, from the initial playback delay to the occurrence of stalling events.

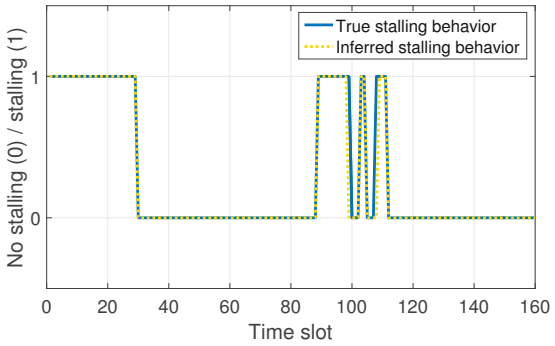


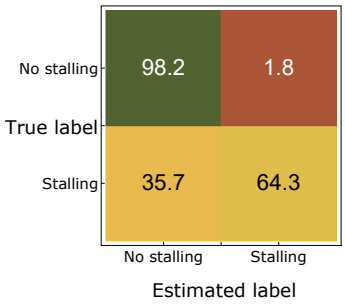
Figure 3.15: Example of ViCrypt real-time stalling detection. © IEEE 2020

Video-Resolution Estimation

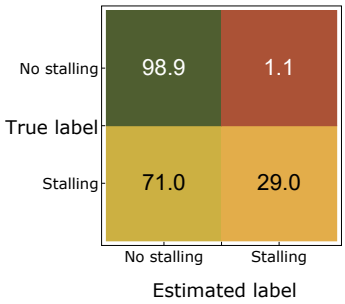
The video resolution is highly linked to the visual quality of the streamed video, and is therefore a crucial QoE metric. The estimation of the resolution is treated as a multi-class classification problem. The considered classes correspond to the typical YouTube video resolutions: 144p, 240p, 360p, 480p, 720p, and 1080p. Thus, the classification problem is based on six classes, which is substantially more precise than other approaches, e.g., [102, 2, 93]. After considering only time slots with a valid resolution and containing traffic, we end up with a dataset including almost 1,160,000 time slots.

	Accuracy (%)	**5-CV time (minutes)**
DT	92	43
RF10	92	2
ADA	68	125
ERT10	90	1
Bagging	95	37
Bayes	42	1
kNN	73	9
NN	58	507
SVM	54	194

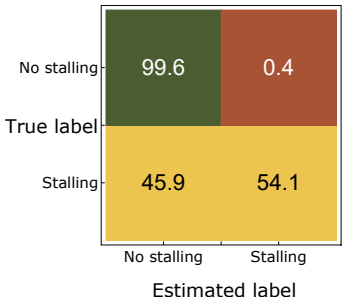
Table 3.4: Benchmarking of different ML models for the resolution estimation. © IEEE 2020



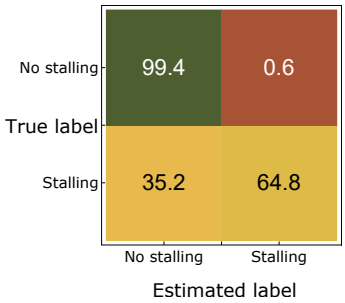
(a) DT confusion matrix.



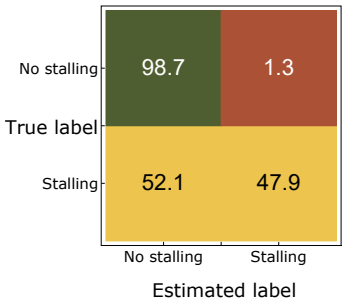
(b) ADA confusion matrix.



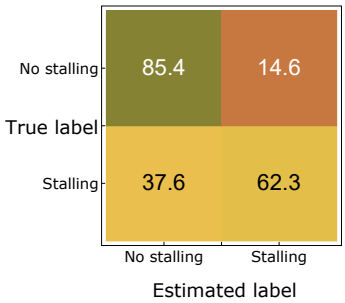
(c) ERT10 confusion matrix.



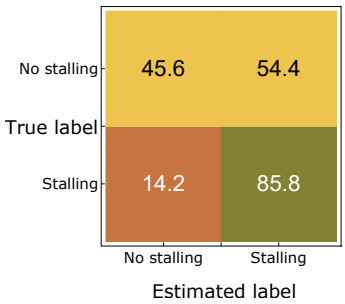
(d) Bagging confusion matrix.



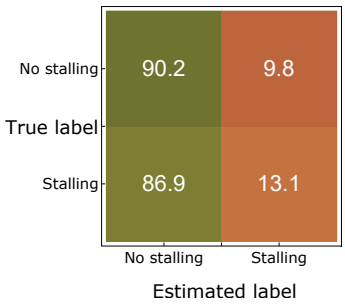
(e) kNN confusion matrix.



(f) SVM confusion matrix.

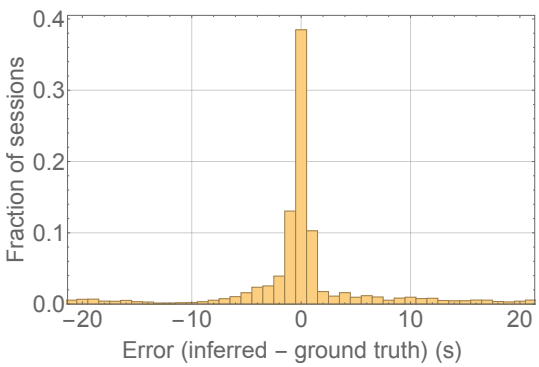


(g) Bayes confusion matrix.

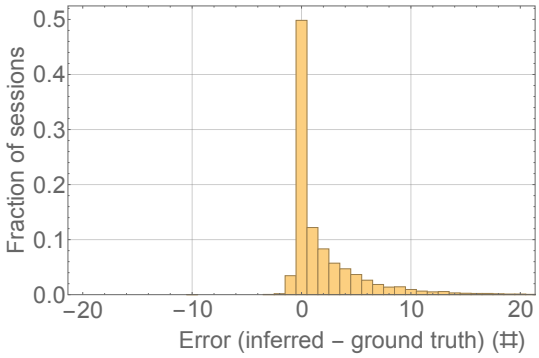


(h) ISO10 confusion matrix.

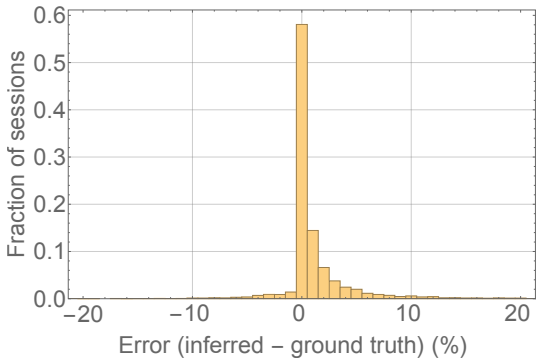
Figure 3.13: Normalized confusion matrices obtained by the benchmarked ML models for the estimation of stalling. The color scale ranges from dark red (poor accuracy) to dark green (excellent accuracy). © IEEE 2020



(a) Initial delay.



(b) Number of stallings.



(c) Stalling ratio.

Figure 3.14: Inference performance for session-based stalling metrics, using ERT10 as the underlying model. © IEEE 2020

We report the accuracy achieved by the different models and the corresponding total processing times for cross-validation in Table 3.4. Except for AdaBoost, all the tree-based methods provide very high overall accuracy, above 90%. kNN also achieves encouraging results, with an accuracy of 73%. The accuracy of Bayes is by far the worst, which is most probably due to its underlying hypothesis that the different features are independent from each other, which does not seem to be satisfied for the video resolution. NN and SVM also yield disappointing results, especially when considering that they needed significantly more time than the other models. Here, we can also verify the benefit of parallelization: besides Bayes, the fastest algorithms are the parallelizable ones, which is a non-negligible advantage for these models. For instance, RF10 and ERT10 were done in at most two minutes, while ADA, NN, and SVM took several hours.

As the video-resolution classes are also strongly imbalanced (cf. Figure 3.11b), we take a closer look at the per-class accuracy (i.e., recall) and precision. Results are depicted in Figures 3.16 and 3.17. We left Bayes out of this analysis because of its poor performance.

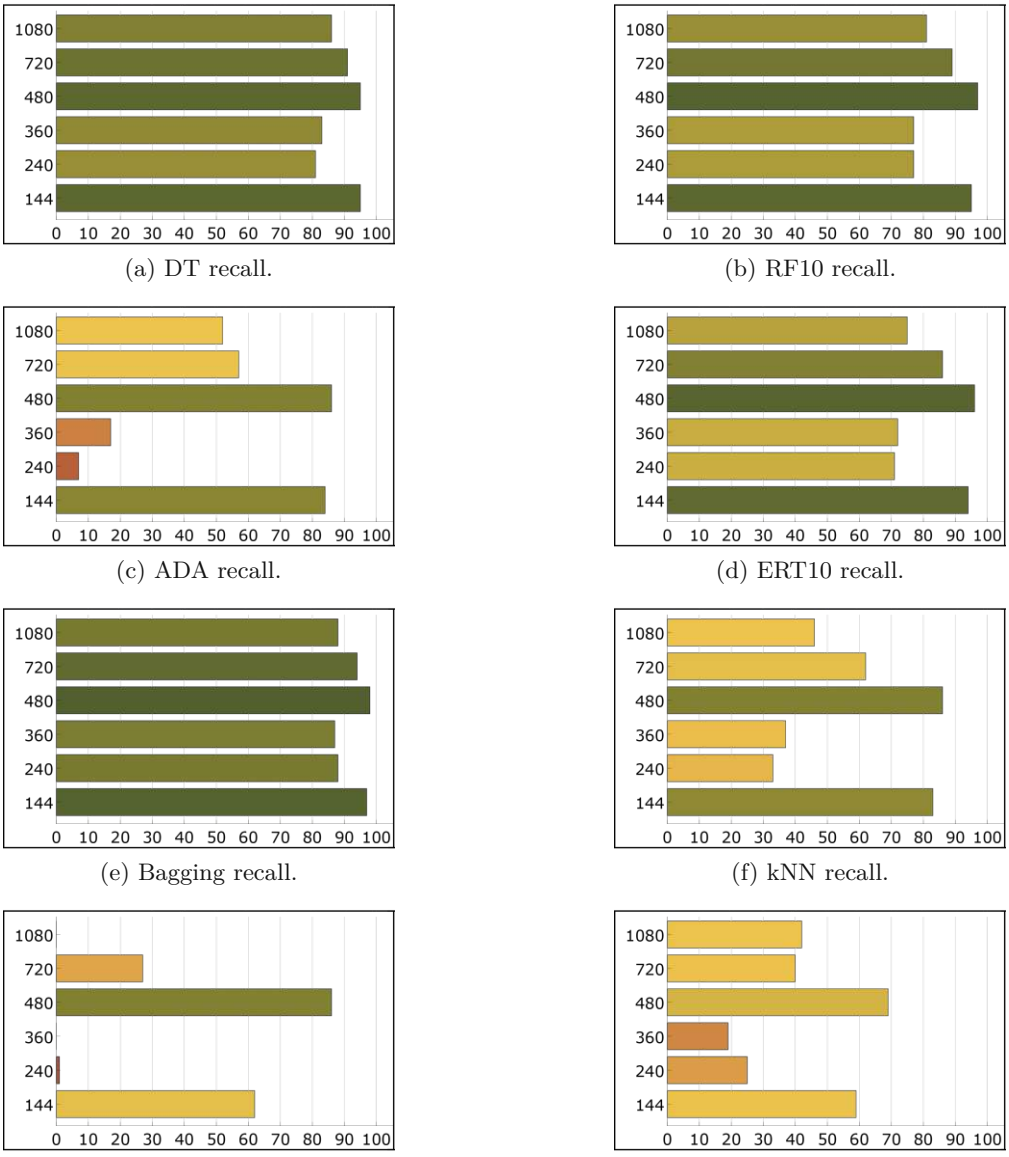
Figure 3.16 reveals that the 480p class is accurately detected by all of the eight mod-

els, with SVM being the worst with an accuracy below 70%. DT, RF10, ERT10, and Bagging achieve a near perfect score for this video resolution. This comes as no real surprise, as more than 50% of the time slots have a resolution of 480p. However, it is interesting to note that most models accurately estimate the 144p class, even though it is a significantly underrepresented class with only 9% of the slots having that resolution. For all the models, these two classes are the ones that are the most accurately detected. For instance, NN obtained an accuracy of more than 60% for 144p and more than 80% for 480p, while for the other classes its accuracy is below 30%. For a couple of models, and especially for ADA and NN, it was challenging to accurately classify the 240p and 360p resolutions; for NN, the accuracy for 360p is even close to 0%. In case of ADA and NN, the two classes were very frequently detected as either 144p or 480p.

Figure 3.17 shows that the precision of the benchmarked models is similar to the recall: it is highest for DT, RF10, ERT10, and Bagging (always higher than 80%), while it is relatively low for ADA, NN, and SVM. Contrary to the recall, the precision is not systematically high for the 144p and 480p classes. Overall, the per-class analysis gives us interesting insights into the performance of the models and indicates that only DT, RF10, ERT10, and Bagging provide consistently excellent estimation throughout all the six video-resolution classes. For example, even though the total accuracy of kNN is decent, it is mostly due to its performance for the 144p and 480p classes.

Results suggest that RF10 is the most appropriate model for the video-resolution estimation task: this model is extremely lightweight, executes fast, and presents an excellent performance, with a recall and precision close to or above 80% for each resolution class. Similar to the stalling-inference results, Bagging is the best model in the benchmark, with recall and precision close to or above 80% for all resolution classes, but using a more complex underlying structure, as reflected by the cross-validation execution times.

Again, for visualization purposes, Figure 3.18 shows the real-time estimation and tracking of the video resolution produced by ViCrypt for an exemplary YouTube video-streaming session using multiple resolution levels (720p, 360p, 480p, and 144p), using RF10 as underlying model.



(a) DT recall.

(b) RF10 recall.

(c) ADA recall.

(d) ERT10 recall.

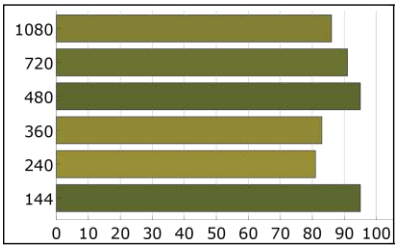
(e) Bagging recall.

(f) kNN recall.

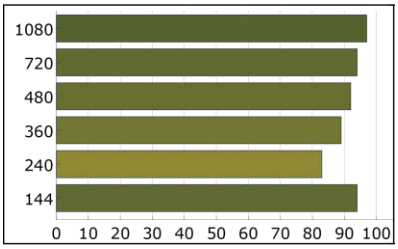
(g) NN recall.

(h) SVM recall.

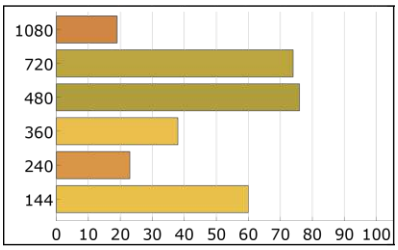
Figure 3.16: Accuracy per class (i.e., recall) obtained by the benchmarked ML models for the resolution estimation. The color scale ranges from dark red (poor recall) to dark green (excellent recall).



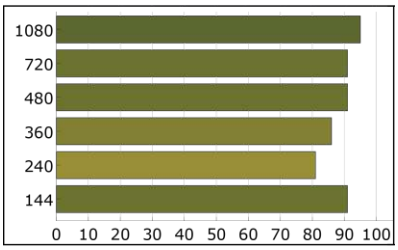
(a) DT precision.



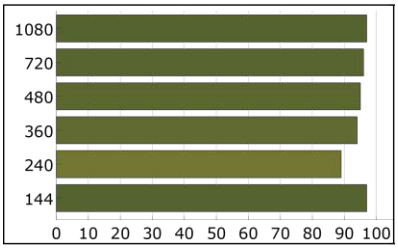
(b) RF10 precision.



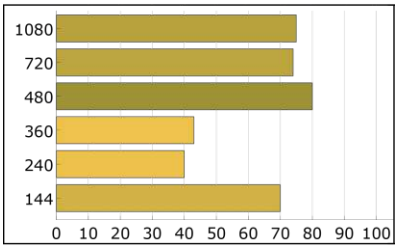
(c) ADA precision.



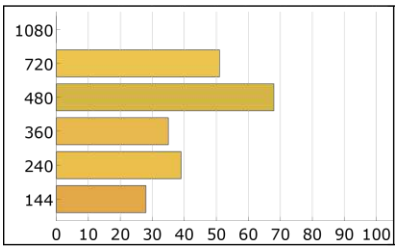
(d) ERT10 precision.



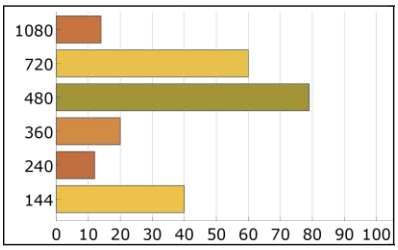
(e) Bagging precision.



(f) kNN precision.



(g) NN precision.



(h) SVM precision.

Figure 3.17: Precision per class obtained by the benchmarked ML models for the resolution estimation. The color scale ranges from dark red (poor precision) to dark green (excellent precision). © IEEE 2020

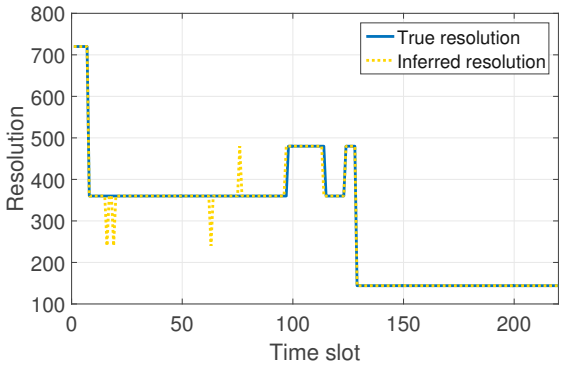


Figure 3.18: Example of ViCrypt real-time video-resolution estimation. © IEEE 2020

Average-Bitrate Estimation

The last estimation target is the average video (encoding) bitrate, which is highly relevant for proactive network management. ViCrypt infers the average video bitrate of the video contents monitored at each 1-second time slot. As the bitrate is per-se continuous, the estimation of the average bitrate is tackled as a regression task. Again, we consider only those time slots with actual traffic and a valid average bitrate label, obtained from the YouTube API. The resulting dataset consists of more than 933,000 samples.

We benchmark the same models as before, using 5-fold cross-validation. The only exception is the Naive-Bayes model, which can only handle classification tasks; we thus replace it by the Bayesian ridge regression (Bayes). For each model, we report the mean absolute error (MAE), the root mean squared error (RMSE), the mean relative error (MRE), and the Pearson linear correlation coefficient (PLCC). As before, we also report the total processing times for the 5-fold cross-validation.

	MAE (kbps)	**RMSE (kbps)**	**MRE (%)**	**PLCC**	**5-CV time (minutes)**
DT	94	246	18	0.88	31
RF10	89	179	18	0.93	36
ADA	492	573	130	0.59	126
ERT10	93	182	19	0.93	7
Bagging	89	179	18	0.93	22
Bayes	2,540	6,530	545	-0.14	3
kNN	229	353	42	0.70	6
NN	333	489	70	0.20	305
SVM	10^{23}	$2 \cdot 10^{23}$	$2 \cdot 10^{23}$	0.12	143

Table 3.5: Benchmarking of different ML models for the average-bitrate estimation. © IEEE 2020

Results are summarized in Table 3.5. We note that the models that worked well for

the video-resolution estimation and stalling detection, namely DT, RF10, Bagging, and ERT10, perform also very well for the inference of the average bitrate. Indeed, these four tree-based models yield the lowest errors, achieving a MAE below 100 kbps, and very high PLCCs close to 1. RMSE and MRE values are relatively low for these algorithms, suggesting that they only rarely make large errors. However, it is interesting to see that RF10 needs significantly more time to process the whole dataset than for the video-resolution estimation. As for the video-resolution inference, Bayes and especially SVM provide disappointing results. With Bayes, ViCrypt obtained a negative PLCC as well as very high error metrics, which underlines the bad performance of the model. With SVM, the system output errors of an unacceptable order of magnitude.

Figure 3.19 depicts the distributions of the inference errors for the different regression models. SVM errors are not reported, as they are simply too large. Overall, the CDFs confirm our observations from Table 3.5. The most promising tree-based methods present errors very close to 0 for a non-negligible fraction of the dataset; this is especially true for DT, which realizes an almost perfect estimation for 60% of the samples. However, DT presents a large RMSE compared to its tree-based competitors, indicating that it yields larger errors than the other tree algorithms. ADA is the only tree-based method where estimation errors are most often quite high, higher than 500 kbps for about 45% of the time slots. Absolute errors are below 100 kbps for approximately 80% of the time slots when using DT, RF10, Bagging, or ERT10 as underlying models.

Based on these results, and again considering the outperformance in terms of computational times, ERT10 seems to be the best algorithm for the estimation of the average bitrate with ViCrypt. Even though error metrics are slightly worse for ERT10 than those for RF10 or Bagging, differences are not significant enough and lightweight models should be preferred.

Finally, Figure 3.20 shows ViCrypt's estimation of the average bitrate for an exemplary video with several bitrate changes. Again, ViCrypt estimates the average bitrate with high precision throughout the whole video. Rather than estimating a too low bitrate, ViCrypt coupled with ERT10 overestimates the ground truth in more than half of the time slots (54%). This overestimation of ERT10 together with the generally low estimation error is advantageous from the point of view of the ISP, as overestimating the video bitrate helps them to avoid allocating insufficient bandwidth in the context of traffic shaping. This could cause the video to stall, which is a major QoE degradation. This behavior could be even enforced by adding a safety margin to the estimations of ViCrypt.

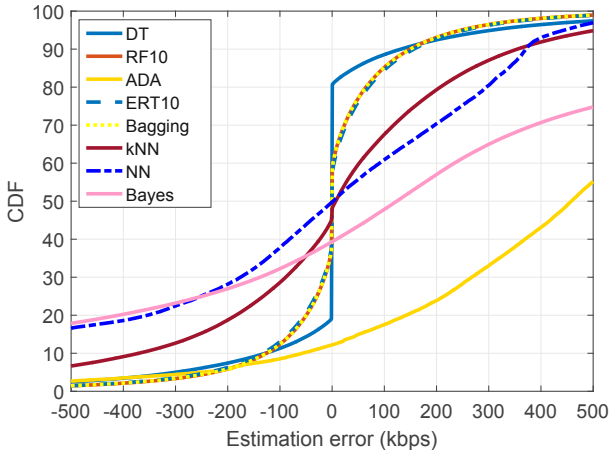


Figure 3.19: Estimation errors (estimated values − ground truth) obtained by the benchmarked ML models for the average video-bitrate inference. © IEEE 2020

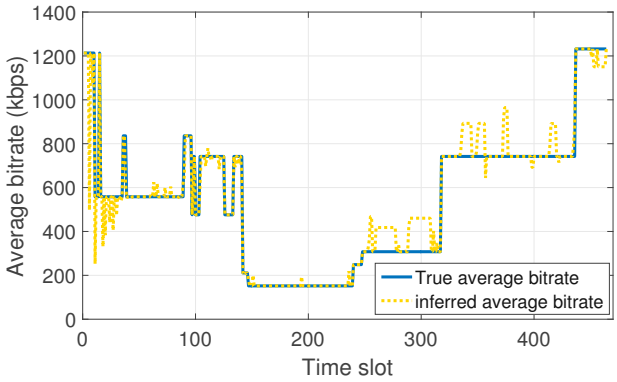


Figure 3.20: Example of ViCrypt real-time average video-bitrate estimation. © IEEE 2020

3.3.6 Feature-Importance Analysis

The results presented so far correspond to ViCrypt models using the full set of 208 features as input for the estimations. In this subsection, we analyze the importance of different feature sets and their impact on inference performance. Using an extensive list of input features is not always the best strategy, as it may negatively impact estimation performance. Using more features increases the dimensionality of the feature space, introducing sparsity issues. In addition, using irrelevant or redundant features may lower model performance in practice. Last but not least, working in higher-dimensional spaces usually results in higher computational times.

We resort to standard automatic feature-selection techniques to identify the most relevant input features for our three stream-based estimation targets. Moreover, we consider

additional feature subsets which might have a significant impact on their own, considering for example the difference between snapshot features – i.e., those computed for the same slot where the estimation takes place – and trend- or session-based features. Based on these guidelines, we divide the full input-feature set into the following six feature subsets:

F_C **subset:** the features representing the *current* time slot, i.e., the time slot for which we want to infer the video resolution (70 features).

F_T **subset:** the features collected for the *trend* window (69 features).

F_S **subset:** the features summarizing the characteristics of the *session* since the beginning of the streaming (69 features).

F_{DOWN} **subset:** the features related to the download traffic for all three windows (81 features).

F_{UP} **subset:** the features representing the upload traffic for all three windows (81 features).

F_{TOP20} **subset:** the 20 most important features, determined using automatic feature-selection techniques (20 features).

To select the 20 most relevant features (F_{TOP20}), we take the best-performing ML algorithm, which is always a tree-based method, and apply an embedded feature-selection technique. For each run of the 5-fold stratified cross-validation, we fit the model on the training folds and detect the 20 most discriminative features. Then, we re-train the model only on those 20 features and test its performance on the test fold. We finally determine the overall top 20 features based on their importance score averaged over the five folds, as well as the average accuracy of the algorithm over the folds with only the selected features.

Before going into the specific performance results achieved with these subsets, let us take a look at the five most relevant features according to the aforementioned feature-selection approach. Table 3.6 reports the five most important features for the three investigated KQIs. For each of the selected features, we additionally report the corresponding temporal window (current, trend, or session) and the importance score. For stalling detection, the most important features come from different subsets. The most important feature for stalling inference comes from the F_T subset (trend window). Nevertheless, almost all of the most important features actually come from the set of session-based features. F_S can be generally considered as the most relevant feature set for stalling estimation, which is in line with previous results in [104], albeit using a different feature-importance metric, namely, the information gain. For video resolution, the top 5 features are all session-related and include statistics about throughput patterns and information related to the inter-arrival times between packets. The results for the average bitrate confirm

	Stalling	Video resolution	Average bitrate
#1 feature	maximum upload packet size (trend) [0.03]	throughput (session) [0.04]	throughput (session) [0.07]
#2 feature	standard deviation of upload packet size (session) [0.02]	burst throughput (session) [0.03]	burst throughput (session) [0.05]
#3 feature	upload volume (session) [0.02]	mean IAT of download packets (session) [0.02]	skewness of upload packet-size distribution (session) [0.04]
#4 feature	standard deviation of download packet size (session) [0.02]	burst throughput of download traffic (session) [0.02]	mean IAT of download packets (session) [0.04]
#5 feature	skewness of upload packet-size distribution (session) [0.01]	coefficient of variation of IAT of download packets (session) [0.02]	download burst throughput (session) [0.04]

Table 3.6: Top 5 most important features for the three stream-based estimation targets tackled by ViCrypt, with their corresponding window (current, trend, or session) and Gini importance scores.

that session-based features are often relevant ones, followed by the features of the trend window and the current time slot.

As we show next in the specific comparison results, session-related features – the F_S subset – are the most important ones, and have the most discriminative power, followed by trend features – F_T. This is coherent with the overall nature of adaptive-video-streaming algorithms, where stronger variations tend to occur at the beginning of the video session, and conditions tend to remain constant over the course of the streaming – as long as the connection remains stable. Features computed for the current time slot generally achieve the lowest importance scores. This suggests that snapshot-like approaches as the one proposed in [2] are less powerful and more prone to overfitting, and probably have poorer generalization capabilities.

Stalling

Table 3.7 reports the estimation performance for stalling in terms of recall and precision for the stalling class, using ERT10 as the underlying model, which is the one we selected in Section 3.3.5 for this task. As before, results correspond to 5-fold stratified cross-validation. Stalling-detection results when using only F_C and F_T subsets are poor for both recall and precision. However, performance dramatically improves when considering F_S features only; indeed, the recall for ERT10 increases from 54% (using all features, cf. Table 3.3) to 72%, and even the precision increases from 88% to 91%, taking the overall accuracy to 99%. This confirms the paramount importance of session-progression

Features	Accuracy (%)	Recall (%)	Precision (%)
All	97	54	88
F_C	96	10	30
F_T	97	17	51
F_S	99	72	91
F_{DOWN}	98	41	87
F_{UP}	98	47	74
F_{TOP20}	97	56	86

Table 3.7: ViCrypt performance for stalling estimation with ERT10, using different feature subsets (overall accuracy and recall/precision only indicated for the stalling class).

Features	Accuracy (%)
All	92
F_C	70
F_T	73
F_S	96
F_{DOWN}	90
F_{UP}	90
F_{TOP20}	95

Table 3.8: ViCrypt performance for inferring the video resolution with RF10, using different feature subsets.

features, which is in line with our above findings and discussion.

When relying exclusively on the 20 most important features, ViCrypt obtains a recall score of 56% and a precision score of 86%, almost on a par with using all features. This tells us the following:

1. The overall statistics describing the entire history of the session are very insightful metrics for detecting stalling.
2. ViCrypt produces highly accurate estimations even with a reduced set of features: instead of using 208 attributes, 69 or even 20 would be sufficient.

This shows that a substantial number of features can be removed with only a minor performance degradation, which even increases the practical applicability of ViCrypt.

Video Resolution

For the case of video-resolution inference, we study the performance of RF10, the model that produces the most promising outcome, based on the different feature groups. The

Features	MAE [kbps]	RMSE [kbps]	MRE [%]	PLCC
All	93	182	19	0.93
F_C	275	407	55	0.58
F_T	253	377	51	0.64
F_S	68	157	14	0.95
F_{DOWN}	105	195	21	0.92
F_{UP}	106	198	21	0.92
F_{TOP20}	81	175	16	0.93

Table 3.9: ViCrypt performance for estimating the average bitrate with ERT10 using different feature subsets.

results in Table 3.8 also reveal that the features of F_C and F_T yield the poorest results in terms of accuracy, indicating that they are insufficient to infer the video resolution with high precision. Indeed, their accuracy is more than 15 percentage points below the accuracy obtained by ViCrypt based on the entire feature set. However, ViCrypt performs very well when used with either F_S, F_{DOWN}, or F_{UP}, with F_S performing even better than the entire feature set. The average accuracy of ViCrypt when using only the top 20 features is highly encouraging: it is equal to 95%, confirming the above finding that a substantial number of features can be removed.

Average Bitrate

Table 3.9 reports the results obtained for the estimation of the average bitrate, using ERT10 as underlying model. The differences in terms of performance between the considered subsets and the whole set of features are much more significant than in the video-resolution case. As a matter of fact, with F_C and F_T, the value of the MAE is nearly three times higher than when relying on the whole feature set; the other error metrics underline the poor performance of these feature sets. In case ViCrypt bases itself on the download- or upload-traffic information, the obtained errors are only slightly higher than when inferring from all the features. However, as for the stalling detection and video-resolution inference, ViCrypt yields excellent results when coupled with F_S features only, showing once again that information about the session history is the most valuable one. Again, using only the top 20 selected features for the ERT10 model yields slightly more precise estimations than when using the whole feature set. Indeed, the MAE and the RMSE decrease to 81 kbps and 175 kbps, respectively.

3.3.7 Practical Considerations for Real-Time Operation & Discussion

Finally, we elaborate on the presented results and discuss the applicability of ViCrypt in practice. To ensure that ViCrypt is scalable and can be deployed in the wild, we analyze several key aspects in terms of computation time, and execute multiple tests on two machines with completely different technical specifications: on *server*, the high-end computer presented in Section 3.3.4, and *laptop*, which includes an Intel Core i5-

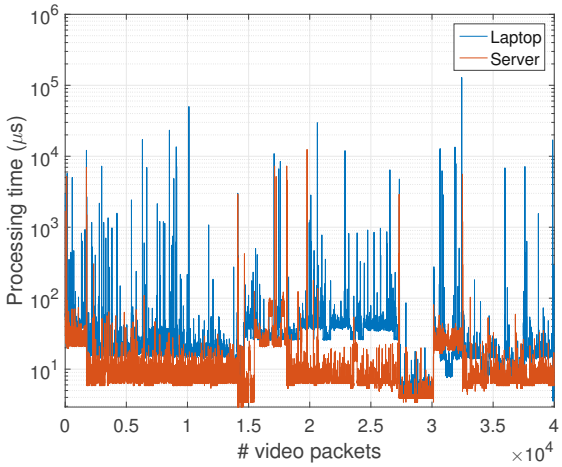
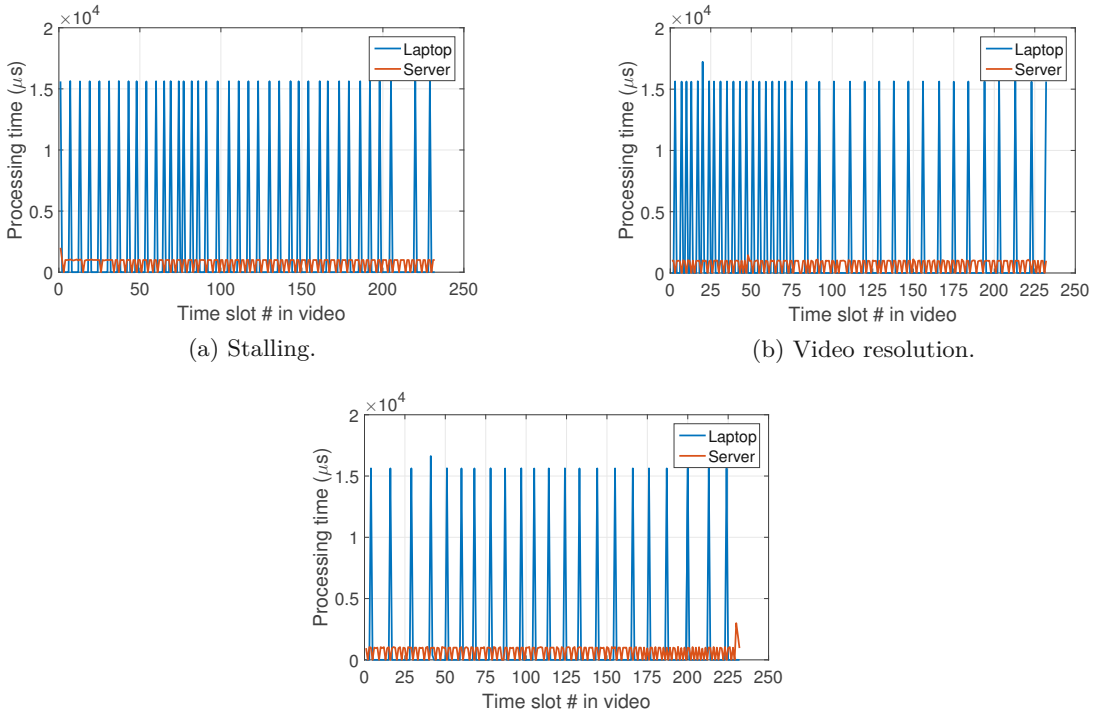


Figure 3.21: Time needed to update the ViCrypt features each time a new packet arrives (log scale). © IEEE 2020

4200U CPU with two physical cores and a total of four virtual ones, eight gigabytes of RAM, and an integrated GPU Intel HD Graphics 4400. We show that ViCrypt runs extremely quickly, with minimal memory footprint. The evaluation is not done at scale, but considering the end-to-end processing of single video sessions. Indeed, the main target of ViCrypt is video-streaming-QoE monitoring at devices installed near the end user (e.g., home routers), where traffic load enables real-time monitoring with limited hardware capabilities.

Feature extraction: to demonstrate the real-time properties of ViCrypt for the feature-extraction process, we record at each packet arrival the time needed to update the feature set for a session lasting four minutes. The results are reported in Figure 3.21. Feature updates are performed extremely quickly on both machines: they take only a couple of microseconds. On *server*, the peak value is about 12 ms, while the average duration is of only 13 μs. More than 90% of the updates took less than 25 μs. Even on *laptop*, the average processing duration is 37 μs, with a maximum value of 129 ms, which is still almost an order of magnitude smaller than the time-slot length of 1 s.

Stalling detection: we measure the time needed by the most suitable model (ERT10) to detect the stalling of a time slot on both *laptop* and *server*. To do so, we train the model on 80% of the data, i.e., on four folds, and record, for one video session in the remaining fold, the time required to detect the occurrence of stalling per time slot. We use the same video as before, and, to make computations harder, we disable the parallelization of the algorithm for the estimation phase. The estimation times for this task are depicted in Figure 3.22a. The model runs very fast, with an average



(a) Stalling.

(b) Video resolution.

(c) Average bitrate.

Figure 3.22: Time needed to estimate video-QoE metrics from features for an exemplary YouTube video session. © IEEE 2020

of 740 μs and a maximum of 2 ms on *server*, and an average of 2.5 ms on *laptop*, which confirms that ViCrypt can also infer stalling in real time.

Video-resolution estimation: we measure the time needed by the best performing model, RF10, for a single estimation of the video resolution, following the same procedure. Figure 3.22b shows the processing times for video-resolution estimation for the consecutive time slots of the exemplary video. On *server*, almost all of the durations are around 1 ms, with an average of 700 μs and a maximum of around 1.4 ms; on *laptop*, the average duration is 2.5 ms and all estimations are available in less than 20 ms. Again, results confirm that ViCrypt performs video-resolution inference very fast, and significantly faster than the time slot length of 1 s.

Average-bitrate estimation: finally, we analyze the time needed by the best model (here, ERT10) to infer the average bitrate of a time slot. The observed estimation times are almost identical to the ones observed for the video resolution on both machines, as we can see in Figure 3.22c, with an average value of 700 μs and a maximum of 3 ms on *server*, showing that ViCrypt can also infer average video bitrate in real time.

	Sessions	Stalling		Video Resolution – P						Video Resolution – R					
		P	R	144p	240p	360p	480p	720p	1080p	144p	240p	360p	480p	720p	1080p
ViCrypt	15,000+	91.0	72.0	96.5	89.5	93.4	95.1	96.2	96.0	96.2	88.3	87.5	98.1	93.5	88.2
Requet	580	70.4	51.9	80.6	68.7	49.2	64.9	60.6	75.0	79.9	64.3	64.4	63.8	54.5	76.9
INFOCOM'18	10,863	80.9	83.9	73.2				80.1		71.1				82.1	

Table 3.10: Referencing performance comparison between ViCrypt, Requet [3], and INFOCOM'18 [2]. Results correspond to numbers reported in [3] and [2], for different datasets. See Table 3.1.

ViCrypt vs. State of the Art

To conclude our study, we provide some indicative results comparing the estimation performance of ViCrypt against the two most similar systems in the literature, namely Requet [3] and INFOCOM'18 [2]. While a re-implementation of both systems for benchmarking purposes is out of the scope of our study, we present in Table 3.10 the performance results reported by the authors of both approaches in the corresponding papers [2, 3]. Naturally, this is not intended as a valid or fair comparison among approaches, as the used datasets are not the same. Still, we decided to include the table to better position ViCrypt within the state of the art, and to serve as reference or baseline for the results presented in this study.

We consider the estimation of two of the KQIs, namely stalling and video resolution, as neither Requet nor INFOCOM'18 are designed to estimate the average video bitrate. In addition, while both ViCrypt and Requet tackle the video-resolution inference problem as a multi-class classification task, using exactly the same resolution levels, INFOCOM'18 considers only a binary classification task, defining *low video resolution* as all resolutions below 480p, and *high video resolution* for levels above 480p.

As already mentioned, the size and heterogeneity of the considered datasets is significantly different for the three systems. A particularly challenging issue for ViCrypt is that our dataset is highly imbalanced, especially when it comes to the occurrence of stalling, with only a small fraction of videos and time slots experiencing stalling. On the contrary, the dataset used in INFOCOM'18 is almost perfectly balanced in terms of stalling, with about 106,000 time slots corresponding to no-re-buffering and 94,000 slots reported as stalling. Unfortunately, this dataset is not publicly available, preventing us from doing further analysis and comparisons.

Nevertheless, Table 3.10 shows that ViCrypt achieves similar or even better results than INFOCOM'18 in the binary detection of stalling events, and that both systems significantly outperform Requet, which presents quite poor results for stalling detection. Here, ViCrypt uses the ERT10 model with only F_S features (cf. Table 3.7), which is still not the best of all models reported in the study – Bagging performance using only F_S features is even better. Regarding the video resolution, ViCrypt provides highly accurate results in terms of precision and recall, while both Requet and INFOCOM'18

report lower performance on the classification task. INFOCOM'18 achieves relatively poor performance for video-resolution estimation, even if the KQI is addressed as a plain binary-classification task.

All in all, we can conclude that ViCrypt is not only able to estimate video-quality metrics with high accuracy – similar to or even potentially better than the state of the art – but also to do it in real time, with minimal computation-time requirements.

3.4 Conclusions

In this chapter, we presented two tools for video QoE monitoring, namely YoMoApp, an Android application for session-based YouTube QoE monitoring, and ViCrypt, a machine-learning-driven system for real-time estimation of QoE-relevant metrics of video streaming.

Through a detailed analysis of the YoMoApp data, we demonstrated that there has been a systematic performance and QoE improvement of YouTube in mobile devices since 2014, additionally evidencing that these enhancements might have a direct impact on the user engagement in YouTube mobile. We also built ML models to estimate key session-based video-QoE metrics relying exclusively on network-layer features easily accessible through the Android API. We obtained very encouraging results, especially with a 10-tree random-forest model. Indeed, random forests provide in general better results in terms of accuracy than other models, with a much smaller computational overhead for building decision trees as compared to neural networks or support-vector machines. The presented models could enable a broader, non-intrusive, and privacy-preserving approach for large-scale, QoE-based monitoring of YouTube mobile, as they could be directly applied as filtering modules running on the mobile device's OS, without accessing any higher-layer metric to perform the estimations.

With ViCrypt, we were able to very accurately estimate KQIs in real time, more specifically with a time granularity of only one second. This is, to the best of our knowledge, the finest granularity so far used for quality inference in the context of encrypted traffic. We benchmarked a wide range of ML models and found that tree-based techniques are the most appropriate algorithms for ViCrypt. We also analyzed the feature importance and showed that ViCrypt performed best with only a reduced feature set. Here, the features summarizing the characteristics of the session since the beginning of the streaming were the most relevant ones. As a last step, we evaluated key computational aspects and demonstrated that ViCrypt runs extremely quickly, with minimal memory footprint. Overall, we demonstrated that our framework is a very powerful tool to infer KQIs of YouTube.

CHAPTER 4

Web-QoE Monitoring and Analysis

Notice

Parts (text as well as figures and tables) of this chapter have already been published in the following publications:

[C6] *Mobile Web and App QoE Monitoring for ISPs – from Encrypted Traffic to Speed Index through Machine Learning*
P. Casas, S. Wassermann, N. Wehner, M. Seufert, J. Schüler, T. Hoßfeld
Wireless and Mobile Networking Conference, October 2021*
DOI: 10.23919/WMNC53478.2021.9619058

[C5] *Are you on Mobile or Desktop? On the Impact of End-User Device on Web QoE Inference from Encrypted Traffic*
S. Wassermann, P. Casas, Z. Ben Houidi, A. Huet, M. Seufert, N. Wehner, J. Schüler, S. Cai, H. Shi, J. Xu, T. Hoßfeld, D. Rossi
International Conference on Network and Service Management, November 2020*
DOI: 10.23919/CNSM50824.2020.9269095

[A2] *How Good is your Mobile (Web) Surfing? Speed Index Inference from Encrypted Traffic*
S.Wassermann, P. Casas, M. Seufert, N. Wehner, J. Schüler, T. Hossfeld
SIGCOMM Posters, Demos, and Student Research Competition, August 2020
DOI: 10.1145/3405837.3411382

The data-collection setup has been developed by the researchers of the university of Würzburg. The feature-extraction process has been developed by Dr. Pedro Casas. The paragraphs describing this setup have mostly been written by this team, even though I edited them for clarity (introduction to Section 4.2, and Section 4.2.3). I mostly focused on the parts about data analytics and ML during our collaboration and wrote the corresponding sections.

The Web is one of the most relevant applications of the Internet. Its performance is highly relevant to the success of every online service, as it severely impacts the engagement and churn of users. The assessment of a Web service as perceived by the end user can be performed through the corresponding Web Quality of Experience (QoE), which is very challenging to measure. Different from other services, such as video streaming or gaming, Web browsing is composed of numerous multimedia components and embedded services; loading a Webpage today requires tens of flows to get the various page resources located in diverse servers from different content providers. Moreover, the expectations and the assessment of the browsing experience itself might significantly vary depending on the type of application that is being used by the end users, and even what portion of the application they are interacting with. For example, the expectation for speed when searching for flights on a travel site is different than it is for loading the checkout page. In this complex process, the network plays a key role in the users' Web QoE, forcing Internet Service Providers (ISPs) to deploy effective means to monitor Web QoE as perceived by their customers.

The literature on Web-performance analysis presents a wide range of objective metrics capturing the performance of Webpage loading, including metrics such as Page Load Time (PLT), SpeedIndex (SI), and Above the Fold Time (AFT). However, all these metrics require access to the application layer, which is hidden from the eyes of the ISP by the wide deployment of end-to-end network traffic encryption.

The analysis of Web QoE from purely in-network, encrypted traffic measurements is still an under-explored problem. We bring the following contributions to the Web-QoE community:

- **Multi-device QoE models:** we consider Web QoE not only for desktop devices, but include smartphone and tablet Web QoE, conceiving cross-device generalizable models. Smartphones represent the lion's share of Internet-access devices today, with nearly three quarters of the world population using exclusively their smartphones to access the Internet by 2025 [112]. According to our results, a model trained only on desktop browsing data provides poor Web-QoE-estimation performance when applied to smartphone and tablet measurements. More generally, a model trained on data coming from a single device yields disappointing results when applied to data coming from any different device.

- **Subjective Web-QoE estimation:** besides training regression models to estimate objective Web-QoE metrics – in particular SI –, we rely on real end-user data from previous studies [113] to build Web-QoE classification models for subjective metrics, as for instance Mean Opinion Scores (MOS).
- **ML benchmarking:** we present an extensive benchmark comparing the performance of different ML models for both estimation tasks – regression for SI inference and classification for QoE-class estimation.
- **Packet and flow-level models:** we conceive models working either at the packet level or at the flow level; the proposed flow-level models achieve highly similar performance to the packet-level ones, but using an order of magnitude fewer input features, which demonstrates that they have strong potential to become practical monitoring solutions.

In Section 4.1, we present an overview of the literature about Web-QoE monitoring and analysis. In Section 4.2, we introduce the generated dataset and the corresponding analyses. In Section 4.3, we benchmark a set of packet-level ML models for QoE inference and classification on desktop devices. We also highlight that these kinds of models do not generalize well to other devices. In Section 4.4, we build a multi-device model, trained on data coming from different types of devices, and show that it achieves high inference performance with a strong generalization potential across desktop, smartphone, and tablet devices. In Section 4.5, we create a multi-device model based on flow-level features instead of packet-level features. Our study reveals that the same performance can be achieved using significantly fewer input features. Section 4.6 concludes this chapter.

4.1 Related Work

Initial Web-QoE models were based on Page Load Times (PLT) [114, 72], and are still broadly used in practice to infer user satisfaction in Web browsing, as for instance in the ITU-T G.1030 model [115]. However, PLT is a poor proxy for user perception of Webpage loading times. Indeed, the actual Web content visible to the user is usually displayed much earlier than all the initially invisible parts, as most Webpages often stretch beyond the browser's viewport. Additional in-browser metrics have been accordingly devised to better characterize the page-rendering progress on the screen. An example is the Above the Fold Time (AFT), i.e., the time until the visible portion of a Webpage has been fully loaded, which has been also tested within traditional Web-QoE models [116]. Newer Web-QoE metrics have been introduced recently, such as the SpeedIndex (SI). Besides single-metric modeling, ML-based approaches have been explored [117, 113] to model Web QoE from a combination of metrics.

Another direction in the literature suggests to understand how external components influence Web QoE. Prior work has studied the impact of network-quality fluctuations and

outages on Web QoE [118, 119]. But besides network quality, other components influence QoE. They are linked to the specific Webpage content – aesthetics [120], usability [121], etc. – as well as to the device type – desktop, smartphone, tablets [122]. These papers show that smartphones and tablets have their own characteristics, not only regarding screen sizes, but also in terms of content rendering and Web designs. Most of these papers focus on Web QoE in controlled, small-scale lab environments.

Others directly rely on in-browser metrics as a proxy to infer Web QoE, conducting large-scale active-measurement campaigns. For example, the impact of multiple features such as transport protocols and network performance on PLT and AFT is studied in [123], based on a set of 244 million measurements collected during six months for the top-10,000 Alexa Websites. Other papers also measured the impact of similar features on PLT and SI or AFT in different countries and different types of networks [124], including mobile ones [125].

Most of prior work stayed at the application level: this situation is problematic for ISPs, who have no direct access to in-browser metrics. In recent years, TLS encryption has even narrowed the information that ISPs can collect from the network side, and previous approaches [126] based on Deep-Packet-Inspection (DPI) and HTTP-traffic analysis are no longer applicable. Other papers [30, 127] developed correlated approximations to the SI metric, such as Byte/ObjectIndex [30] and Pain Index [127], which can be computed from statistics of packet- and flow-level measurements, thus seamlessly operating with encrypted traffic.

When it comes to the specific case of Web-QoE monitoring in mobile devices, there have been multiple papers using machine learning [96, 128, 129, 130] or simple modeling approaches [131] to map application-layer [129, 130] or network-QoS metrics [96, 128, 131] into QoE-related metrics. From these, two papers [96, 128] are the closest to our work, but both offer analysis approaches which are no longer applicable due to HTTP-traffic encryption [128], or do not address the specific problem of Web browsing [96].

4.2 Web-QoE Datasets & Modeling Approach

Note about adoption of text from co-authors: The text in the introduction of Section 4.2 up to 4.2.1 was mainly contributed by co-authors to the joint paper [C5], even though I edited it for clarity.

The proposed solution to the Web-QoE monitoring problem consists of training supervised ML models to map network-traffic features, extracted from the encrypted network-Webpage traffic, into relevant Web-QoE metrics. The approach is data-driven, and thus needs datasets containing both the collected traffic traces – the *input* – and the targeted Web-QoE metric – the *ground truth*. The diagram presented in Figure 4.1 presents the workflow and the different stages of the solution.

To fully control the generation of such datasets, we built a measurement testbed based on multiple private instances of WebPageTest (WPT) [1], an open-source Web-performance-

[1] https://www.webpagetest.org/

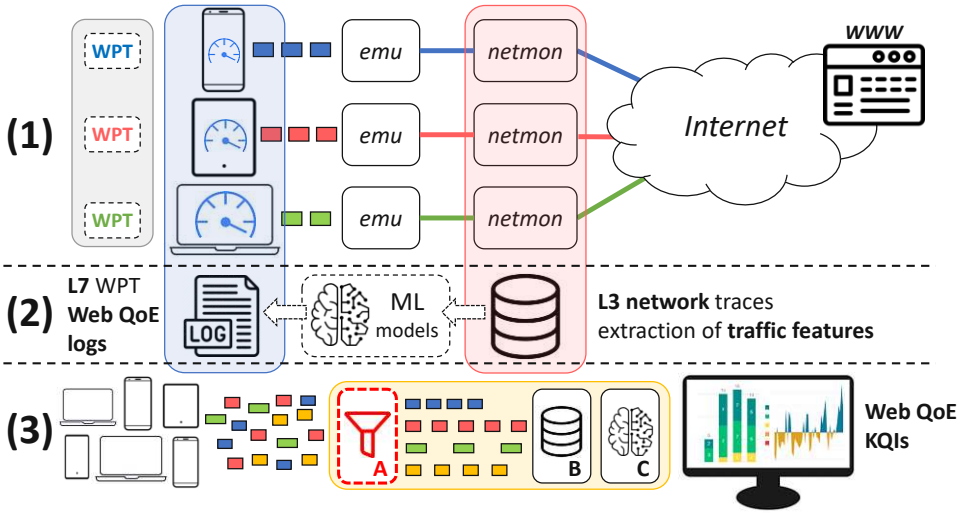


Figure 4.1: Diagram and workflow of the proposed solution.

analysis tool commonly used both in industry and academia. Rows **(1)** and **(2)** in Figure 4.1 depict this testbed and its usage. Different from previous studies [123, 132, 30, 125, 124, 133], which studied Web QoE exclusively for desktop browsers and desktop devices (or in some exceptional cases, emulating mobile devices), our measurement testbed consists of three different, non-emulated types of devices, including a smartphone (Google Pixel 2 XL), a tablet (Google Pixel Slate), and a desktop computer (laptop), using WPT agents for Android and Linux. Chrome (the last stable version) is used as browser. Instead of leveraging in-device WPT traffic-shaping capabilities, devices are connected to the open Internet through independent network emulators (*emu*), which makes it possible to have more realistic network-access-performance configurations in terms of bandwidth, latency, packet-loss rate, etc. This allows for heterogeneity in the generated measurements. Configurations used in the study include access-downlink bandwidth up to 10 Mbps, packet-loss rates up to 10%, and RTTs up to 100 ms. Using WPT measurements, the platform extracts about 90 different KPIs and Web-QoE metrics, indicated as L7 Web-QoE logs in Figure 4.1, row **(2)**, such as PLT, SI, AFT, ByteIndex [30], and Time to Interactive (TTI), as well as content characteristics of the visited pages. Network traffic is captured at an intermediate passive monitoring device (*netmon*) and stored as `.pcap` traces, from where model input features are extracted, indicated as L3 network-traffic features in Figure 4.1, row **(2)**.

For this study, we generated a per-device-type balanced dataset of more than 40.000 Webpage *loading sessions* (i.e., the loading of a single page), targeting the top 500 Websites according to the Alexa top-sites list [2]. The same pages are visited multiple times for each device type, using the same access-network setups. We do not consider the effect of caching, i.e., tests correspond to a first-view loading session. As it has been

[2] `https://www.alexa.com/topsites`

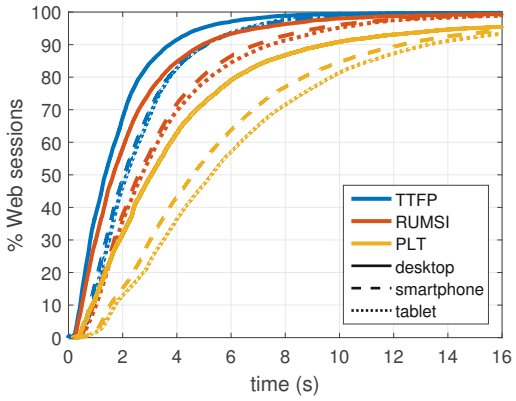


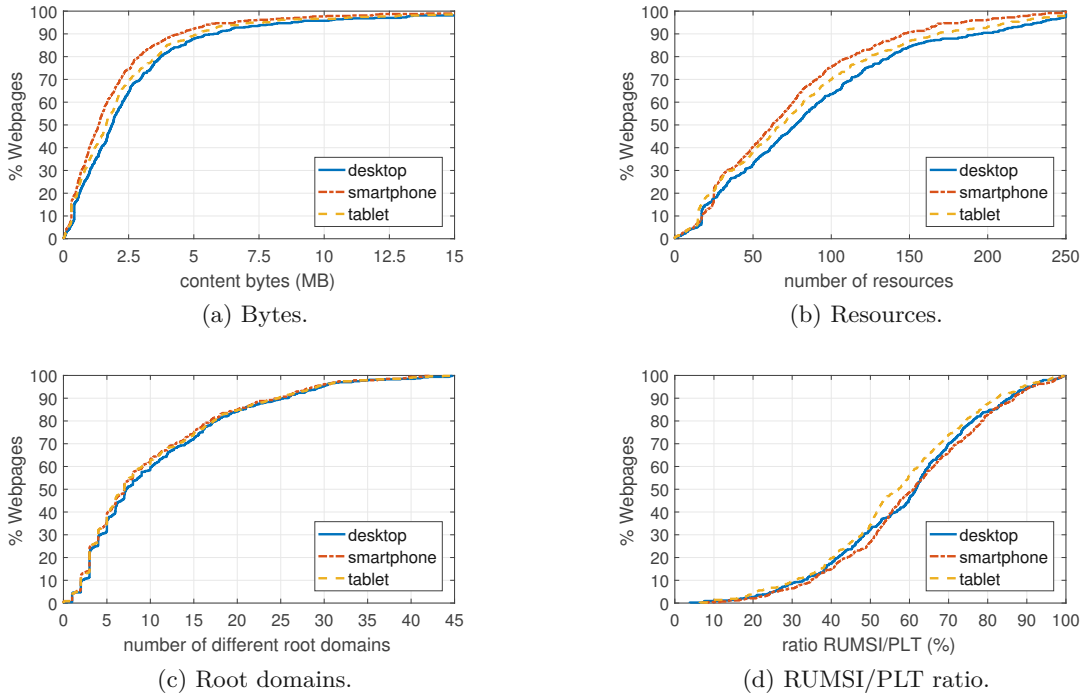
Figure 4.2: Distribution of time performance metrics (TTFP, RUMSI, and PLT) per device.

shown in [132], while caching has an impact on the performance of inference models, this impact is limited, and models generalize well across different protocol and caching settings. We focus on the inference of one particular Web-QoE metric, the SI, which is today one of the most accepted metrics reflecting Web QoE. Nevertheless, the methodology applies to any other similar Web-QoE metric. As shown in [30], measuring the SI proves to be cumbersome in terms of computational resources and might introduce bias in the data capturing/processing, mainly due to the video capturing and analysis. This is particularly critical on smartphones and tablets, which are generally resource-constrained; therefore, instead of focusing on the SI metric, we collect the Real User Monitoring SpeedIndex (RUMSI).

Finally, we assume that the measurement system takes as input network traffic from single Web sessions. In an operational deployment in the wild – see row **(3)** in Figure 4.1 –, the traffic mix of concurrent Web sessions must first be disentangled (step **A**) to then extract features from the traffic belonging to each Web session (step **B**) and apply the trained model(s) (step **C**). Our study exclusively addresses steps **B** and **C**, as we run our sessions sequentially. Nevertheless, in case of concurrent Web sessions, a classification methodology from the literature [127] could be applied to disentangle them.

4.2.1 Data Characterization

The list of top-500 Alexa pages is very diverse in terms of contents and, as we show next, the type of device being used has a visible impact on Webpage characteristics and timing performance. Figure 4.2 depicts the distribution of three relevant Web-QoE metrics, including the Time to First Paint (TTFP), the RUMSI, and the PLT. The values are significantly higher for both smartphone and tablet devices as compared to desktop devices, pointing to a more complex rendering process in mobile devices. This is most



(a) Bytes. (b) Resources.

(c) Root domains. (d) RUMSI/PLT ratio.

Figure 4.3: Cumulative density function, per device type, for page size, number of resources, number of root domains, and SI/PLT ratio.

probably linked to the specific hardware limitations of smartphones and tablets, as well as the particular characteristics of the OS and browser combination – native Chrome in Android. In addition, the way pages are optimized and rendered in mobile devices impacts loading times. Worse loading performance in mobile devices is a commonly known issue in practice[3]. Interestingly, when comparing Android devices, TTFP values are almost identical for smartphone and tablet, RUMSI is slightly higher for tablet, while PLT is significantly higher for tablet. As we see next, this is most probably explained by the fact that Webpages have more content to load in tablets. It is also interesting to note how PLT significantly overestimates the perceived loading time of Webpages, represented by the (RUM)SI metric.

Figure 4.3 characterizes the top-500 Webpages per device type, in terms of (a) page size, (b) number of resources (total number of images, fonts, video, CSS, JavaScript, etc.), (c) number of root domains, and (d) RUMSI to PLT ratio. The latter reflects the complexity of the Webpage in terms of visible content (SI, approximated by the RUMSI) and full content downloading (PLT). As expected, Webpages browsed on desktop devices are

[3] https://backlinko.com/page-speed-stats

heavier than those browsed on smartphones or tablets, which are optimized for smaller screen sizes. The average page size is 2.7 MB in desktop, 2.4 MB in tablet, and 2.1 MB in smartphone.

Figures 4.3b and 4.3c further illustrate the richness and complexity of the Webpages in terms of number of embedded contents and their location at different root domains, with more than 30% to 35% of the Webpages consisting of more than 100 resources, and about 40% of the Webpages fetching resources from more than 10 different root domains. The screen size probably plays a key role in influencing page characteristics, as the number of resources is higher for desktop, followed by tablet, and finally by smartphone. The RUMSI/PLT ratio not only shows how large the overestimation introduced by PLT is in terms of perceived page load times, but also how different this is for the different Webpages. Indeed, about 10% of the pages have a ratio below 30% (the visible content loads significantly faster than the full content) and less than 5% of the pages have a ratio above 90% (the visible content basically corresponds to the full Webpage content).

Our analysis shows that, while most of the analyzed pages are very similar for every device type in terms of size and retrieved contents from external servers, differences can be significant for a small share of the pages. In terms of performance, we see that loading times on Android devices (smartphone and tablet) are significantly higher than on desktop, a common trend observed in practice of Webpage speed analysis.

4.2.2 Subjective QoE Analysis

While the SI is a good objective metric that reasonably captures the Web QoE of real users [116], we resort to previous subjective Web-QoE studies to better understand the expected QoE for the generated dataset. In particular, prior work [113] conducted a subjective study where about 240 participants rated their browsing experience – loading of individual pages using a desktop browser –, according to a 5-level Absolute Category Rating (ACR) MOS score (*bad* QoE being 1 and *excellent* being 5). We rely on their publicly available dataset to identify QoE-related timing thresholds which could translate the RUMSI in our dataset into broad QoE classes. In Figure 4.4b, we depict the relationship between SI and MOS scores obtained in that study. While the SI metric was not directly measured, additional metrics such as the ByteIndex were computed, which can be used as good proxies to the real SI [30].

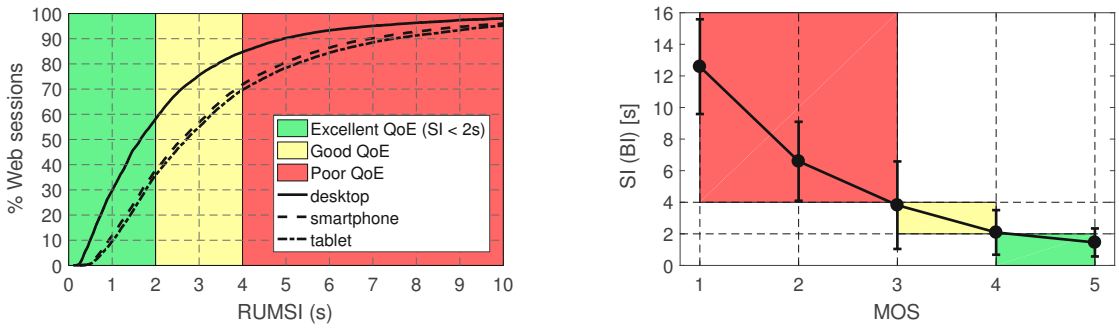
We define three Web-QoE classes:

- **(e)**xcellent: MOS ≥ 4
- **(g)**ood: MOS $= 3$
- **(p)**oor: MOS ≤ 2

This results in SI thresholds of 2 and 4 seconds. Interestingly, the SI thresholds recommended in the industry as target for excellent Web performance vary between 1 second

device	excellent	good	poor
desktop	58%	26%	16%
smartphone	38%	34%	28%
tablet	36%	34%	30%

Table 4.1: QoE-class distribution per device type.



(a) RUMSI and QoE classes.

(b) SI vs. MOS scores.

Figure 4.4: Distribution of (a) QoE classes per device type, based on (d) real-user MOS scores.

(desktop) and 3 seconds (mobile), which are in line with the proposed higher QoE-class threshold of 2 seconds. It is important to note that our thresholds are derived for the case of browsing on desktop devices, and one would expect higher thresholds for Web QoE in mobile devices. Nevertheless, for this study, we assume the same thresholds apply to the three device types. Figure 4.4a shows that about 60%/40% (desktop/mobile) of the loading sessions correspond to excellent QoE, 25%/35% to good QoE, and the remaining 15%/30% result in poor QoE. Table 4.1 provides more details about the exact distribution for each device.

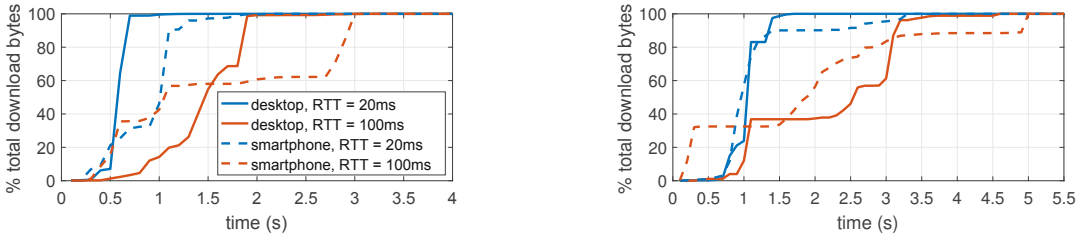
4.2.3 Targets and Input Features

Note about adoption of text from co-authors: The text in Section 4.2.3 was mainly contributed by co-authors to the joint paper [C5], even though I edited it for clarity.

We implemented our Web-QoE monitoring solution with two different inference tasks:

1. Inference of the RUMSI metric, which corresponds to a regression task
2. Estimation of the Web-QoE class {e,g,p}, which corresponds to a 3-class classification task.

We use the same input features in both tasks, derived from the stream of encrypted packets. To define input features, we follow the rationale behind the computation of the



(a) `https://zoom.us` (b) `https://cnn.com`

Figure 4.5: Examples of CBD features or loading curves, using ΔT = 100 ms, and different RTT setups.

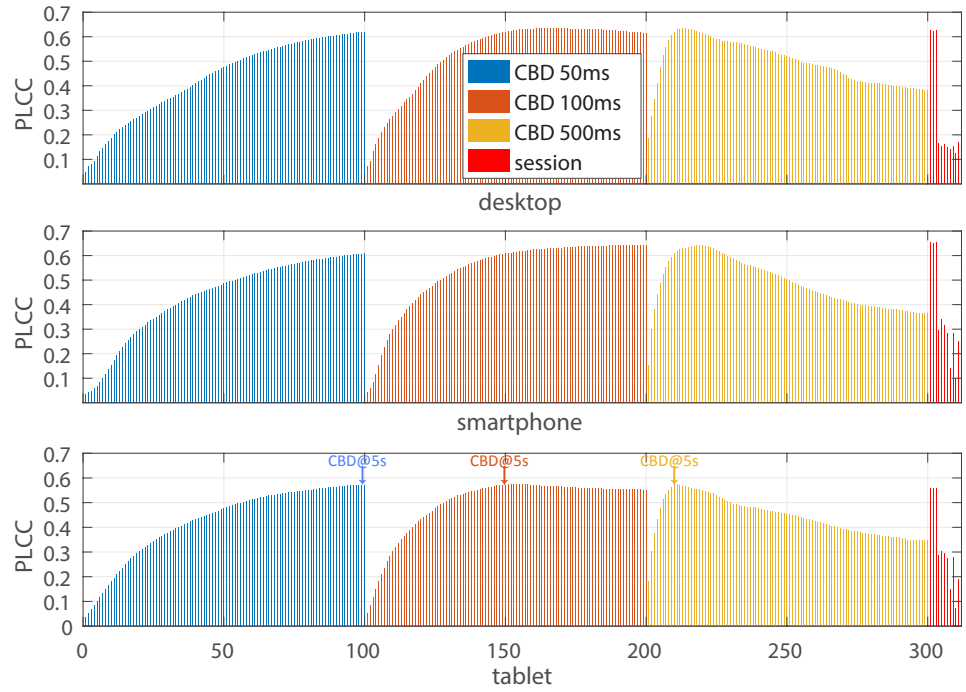
SI metric itself, which considers the whole progress of the page loading. We define the Cumulative Bytes Downloaded features $CBD(i)_{\Delta T}$ as the cumulative number of bytes downloaded from the first collected byte at time t_0 up to time $t = t_0 + i \cdot \Delta T$, with $i = 1, \ldots, m$, normalized by dividing the values by the maximum observed number of bytes, $CBD(t_0 + m \cdot \Delta T)_{\Delta T}$ (such that the maximum value is equal to 1). The CBD features track the download progress of the page bytes, using a time resolution ΔT. Figure 4.5 depicts examples of CBD features for different network configurations, both for desktop and smartphone devices, using $m = 100$ and $\Delta T = 100$ ms. Pages loading faster have a CBD *loading curve* rising sharper and arriving to full loading earlier.

For this study, we took $m = 100$, and three different resolutions for the computation of features, using $\Delta T = 50$ ms, 100 ms, and 500 ms, for a total of 300 CBD features. Using different resolutions helps capture different phenomena in the downloading progress, which potentially impact the SI; it also allows to track different page-load durations, in this case up to 5, 10, and 50 seconds, respectively.

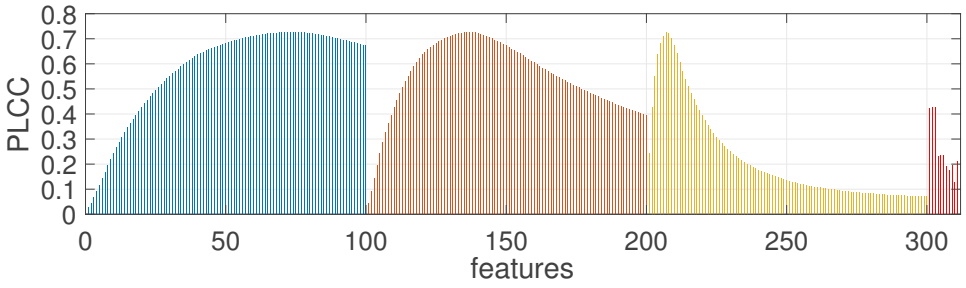
We consider $n = 11$ additional input *session* features, related to the complete page-loading session; these include: full session duration (first to last packet), downlink/uplink session duration (first to last packet in downlink/uplink direction), total number of packets downlink/uplink/full, total number of bytes downlink/uplink/full, and session mean throughput downlink/uplink.

The complete list of features can be found in Appendix C. While these are mostly packet-level features, we extend in Section 4.5 the study to the implementation of flow-level features, achieving highly similar results.

Figure 4.6a depicts the linear correlation (more precisely, the Pearson Linear Correlation Coefficient) between these input features and the RUMSI metric, for different device types. Correlation values are rather high for all devices, with stronger correlations observed for CBD features between 5 seconds and 10 seconds, as well as for session-duration features. Figure 4.6b shows correlation values for the QoE classification problem, for all devices together. Based on the considered time thresholds, higher correlations are observed between 2 and 5 seconds.



(a) Correlation between input features and RUMSI.



(b) Correlation between input features and QoE class.

Figure 4.6: Correlation between each of our 311 input features (300 CBD and 11 session-related) and inference targets.

model	MAE – mAE (ms)	MRE – mRE (%)	PLCC
DT	766 – 288	37 – 18	0.817
ERT10	**598 – 260**	**31 – 16**	**0.879**
RF10	602 – 262	31 – 16	0.860
RF100	**564 – 249**	**30 – 15**	**0.885**
Bagging	600 – 266	31 – 16	0.857
Boosting	767 – 426	48 – 26	0.861
Bayes	976 – 491	64 – 29	0.727
kNN	940 – 496	54 – 29	0.752
XGB	774 – 429	48 – 26	0.849

Table 4.2: Benchmarking of different ML models for RUMSI inference, for desktop.

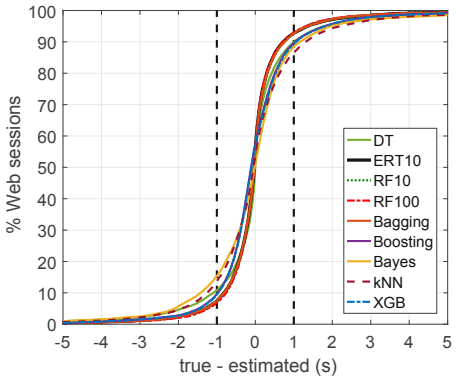
4.3 Desktop Models' Lack of Generalization

We now focus on the Web-QoE estimation tasks. As a reference for performance evaluation, we start by training and evaluating different ML models on desktop measurements. Recall that desktop measurements represent the most common data source so far used in the Web-QoE literature [123, 132, 133, 30, 124, 125]. We then apply the trained models to both smartphone and tablet data to demonstrate the lack of cross-device generalization and poor performance achieved by single-device models in such multi-device scenarios.

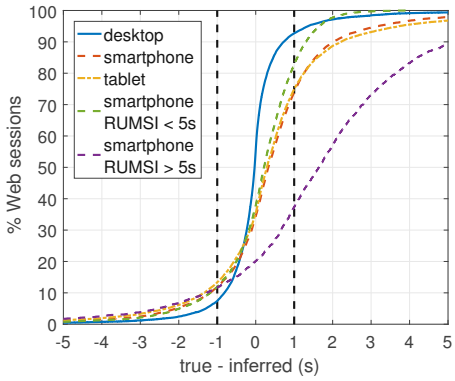
4.3.1 RUMSI Inference on Desktop

Table 4.2 reports the RUMSI-inference performance achieved by nine different ML models, most of them being based on decision trees. The tested models include:

- Single decision tree (DT)
- Multiple types of ensembles using different numbers of trees, such as:
  - Ensemble with 10 extremely randomized trees (ERT10)
  - Random forest with 10 and 100 trees (RF10 and RF100)
  - Bagging with 10 trees (Bagging)
- Boosting:
  - Gradient boosting with 100 trees (Boosting)
  - XGBoost with 100 trees (XGB)



(a) RUMSI-inference performance on desktop measurements.



(b) RUMSI-inference performance on all measurements.

Figure 4.7: Cross-device QoE classification performance. Models are trained on desktop measurements.

The list is completed by Naive Bayes (Bayes), and by the k-nearest-neighbors algorithm (kNN) with $k = 5$, this hyperparameter being calibrated through grid search.

We assess their performance using three commonly used evaluation metrics for regression problems, namely the absolute error (AE), the relative error (RE), and the linear correlation (PLCC). We take both mean (M) and median (m) values for the error metrics, the latter being a robust estimator of the average against outliers. The results correspond to 10-fold cross-validation and all the reported performance metrics are computed over the ten folds. Figure 4.7a additionally depicts the cumulative distribution of the estimation errors.

RF100 attains the best inference performance, with a median absolute error of 249 ms, and a median relative error of 15%. Absolute inference errors are below 500 ms for more than 70% of the sessions. More than 85% of the RUMSI values are inferred with an error below one second. Similar performance is realized by smaller ensembles, e.g., RF10, ERT10, and bagging, which use 10 instead of 100 trees as in RF100. Given the training-speed improvements obtained with the ERT10 model, we take it as the underlying estimation model in subsequent evaluations.

4.3.2 QoE Classification on Desktop

Table 4.3 reports the classification-benchmark results obtained for desktop through 10-fold cross-validation. As for the regression task, the reported values are computed over the folds. Again, RF100 provides the best results, with an overall accuracy (ACC) close to 87%. Recall (R) and precision (P) are above 90% for the excellent QoE-class estimation, but good and poor QoE classes tend to be confused by the estimator. Nevertheless, recall and precision are close to 80% for these classes.

model	desktop						
	ACC	R{e}	R{g}	R{p}	P{e}	P{g}	P{p}
DT	80.3	88.8	66.1	72.8	88.4	66.2	73.8
ERT10	**84.4**	**93.1**	**70.5**	**75.6**	**89.6**	**73.7**	**81.7**
RF10	84.6	93.1	71.6	74.6	90.1	73.1	82.1
RF100	**86.9**	**93.3**	**77.4**	**79.1**	**92.3**	**76.2**	**84.7**
Bagging	85.7	93.2	74.3	76.8	90.8	74.6	84.8
Boosting	82.9	91.1	70.3	73.7	90.1	69.1	79.3
Bayes	60.1	93.0	11.6	19.5	63.2	38.3	46.7
kNN	74.9	87.3	55.1	62.1	81.8	59.1	72.0
XGB	82.2	90.9	69.0	72.1	89.8	67.9	77.6

Table 4.3: Benchmarking of different ML models for Web-QoE-class estimation on desktop. The three levels of QoE correspond to **excellent {e}**, **good {g}**, and **poor {p}** QoE.

4.3.3 Lack of Generalization for Mobile Devices

Now that we have built the models for desktop, a natural question is: *how good would these models perform on data collected from other device types?* This is critical in practice, as a significant, and ever growing, share of Web-browsing activity comes from mobile devices. Figure 4.7b depicts the distribution of inference errors per device type, using the ERT10 model, trained exclusively on desktop data. Table 4.4 summarizes the corresponding performance metrics. There is a strong degradation in estimation performance when applying the desktop model to both smartphone and tablet data. Median absolute errors almost triple as compared to the performance on the desktop dataset. The distribution of errors shows that the desktop model tends to underestimate the RUMSI metric when applied to other devices not present at training time. One could argue that the RUMSI on smartphone and tablet is significantly higher than on desktop (cf. Figure 4.2), thus the performance degradation could be linked exclusively to this mismatch. However, the degradation is also significant when considering only smaller RUMSI values, with median errors more than doubling for the example case of smartphone – from 260 ms to 592 ms –, testing only for RUMSI values below five seconds.

Figure 4.8 shows how the aforementioned cross-device training and validation difficulties also hold when considering different device types when inferring the RUMSI. More precisely, we consider all the possible pairs of training sets and test sets to give the broadest picture of cross-device issues: all combinations of the three device types are explored. The figure reports the usual AE, RE, and PLCC metrics arranged as a training/testing matrix, where rows correspond to the device-type data used for training and columns to the device-type data used for testing. While specialization improves inference per-

device	MAE – mAE (ms)	MRE – mRE (%)	PLCC
desktop	598 – 260	31 – 16	0.879
smartphone	1245 – 721	41 – 28	0.728
tablet	1434 – 724	44 – 28	0.618
smartphone RUMSI < 5s	867 – 592	43 – 29	0.455
smartphone RUMSI > 5s	2812 – 2000	32 – 27	0.667

Table 4.4: Inference performance per device type. The ERT10 model is trained using desktop data.

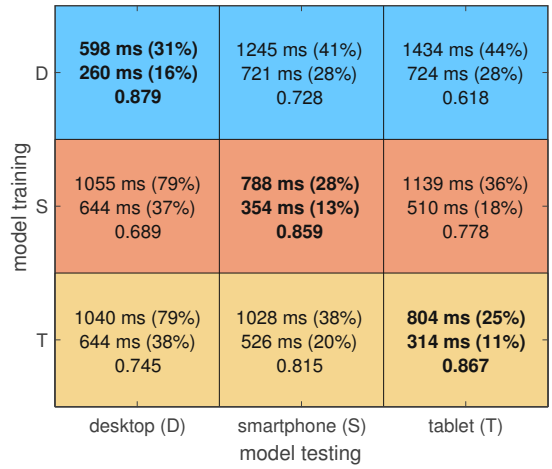


Figure 4.8: Cross-device RUMSI-inference performance, using per-device ERT10 as underlying model. In each cell, the first line gives the means and the second one the medians of both the absolute error (MAE and mEA, in ms) and the relative error (MRE and mRE, in %); the third line is the correlation coefficient (PLCC).

formance – the matrix diagonal –, training a model on measurements from a particular device type and applying the resulting model on measurements from a different device type results in poor inference performance, for all device-type combinations. This lack of cross-device generalization ability also applies to the two types of mobile devices: even though tablets and smartphones are quite similar in terms of characteristics (cf. Figures 4.2 and 4.3), training on a mobile device and testing on another yields poor performance compared to training and testing on the same type of device. However, the difference in performance is not as large as in the case where training happens on a mobile device and testing on a desktop device (or vice-versa).

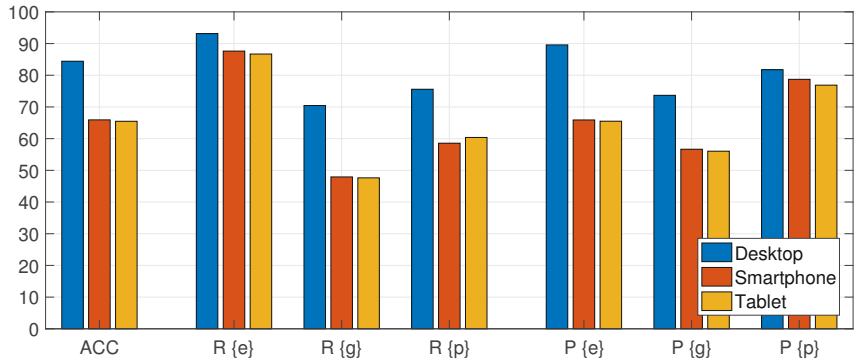
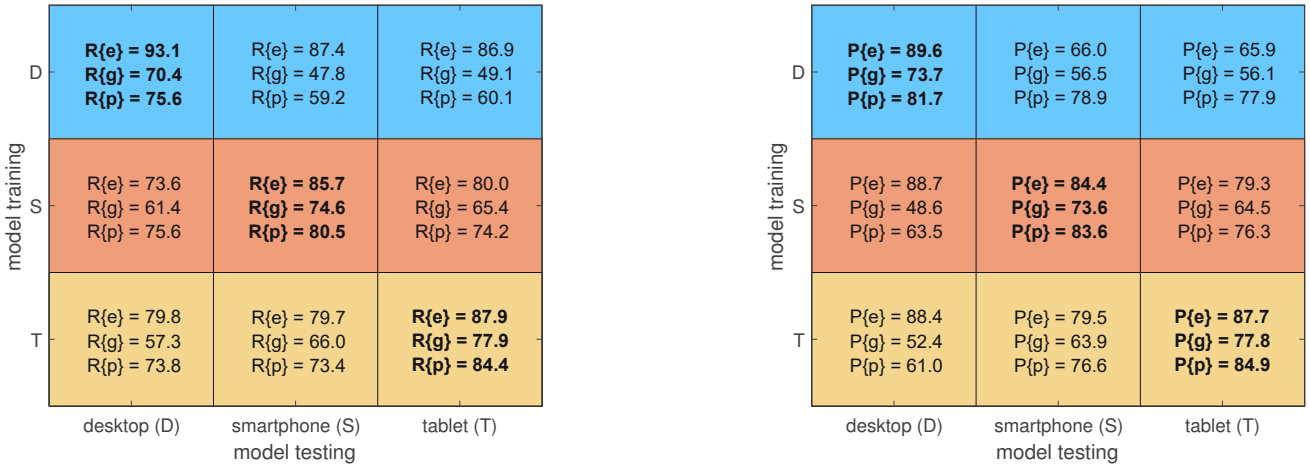


Figure 4.9: Cross-device QoE-class-inference performance, using per-device ERT10 trained on desktop data as underlying model.



(a) Recall.

(b) Precision.

Figure 4.10: Cross-device QoE-class-inference performance across all device types, using per-device ERT10 as underlying model.

Performance degradation is also significant for the QoE-classification problem. Figure 4.9 reports the classification performance per device type, again using the ERT10 model, trained on desktop data. The overall classification accuracy drops from 84% on desktop to 67% on smartphone and tablet. Recall for excellent QoE degrades only slightly, but strongly for the other classes, and most importantly, precision for excellent QoE also drops drastically. Similarly to the RUMSI-inference analysis, we summarize the cross-device training and validation results across different devices in Figure 4.10. We observe the same lack of cross-device generalization, and once again also for the two mobile devices.

device	MAE – mAE (ms)	MRE – mRE (%)	PLCC
desktop	649 – 300	37 – 18	0.874
smartphone	804 – 363	27 – 14	0.855
tablet	798 – 318	25 – 11	0.868
all	750 – 327	30 – 14	0.869

Table 4.5: Multi-device RUMSI-inference performance.

Overall, our study shows that RUMSI inference and QoE-class estimation can be carried out properly using CBD and session-based features, extracted directly from the stream of encrypted bytes. However, models so far proposed in the literature for single device types [123, 132, 30, 125, 124, 133] might not perform properly in the wild, where other devices than desktop machines are used for Web browsing.

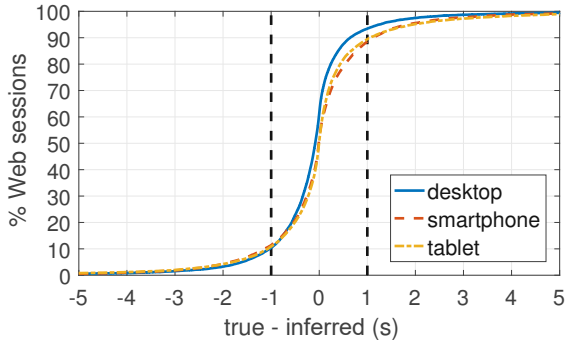
4.4 Multi-Device Models

Now that we have highlighted the lack of generalization and the cross-device issues introduced by per-device models, we take the most natural step to conceive multi-device Web-QoE models. Possible approaches include the usage of stacking/ensembles of specialized models [134] or the inclusion of a pre-processing device-type classification task, preceding the main inference task. However, the simplest approach, exposing models to data coming from all types of devices at training time, already provides high accuracy: these models generalize well across devices, which has high practical appeal as it simplifies deployment.

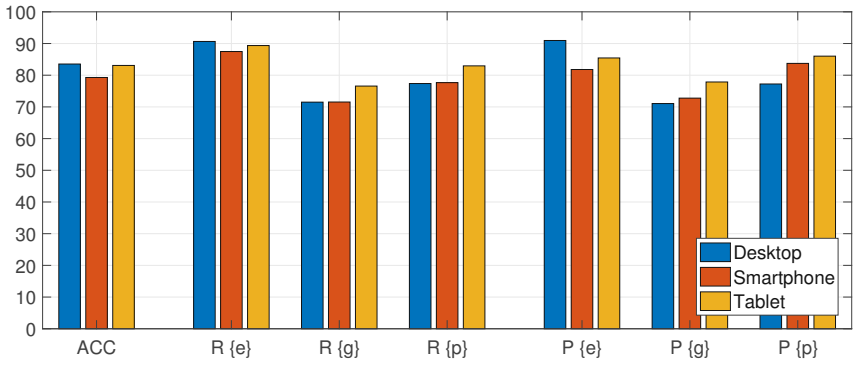
In the context of our study, we apply stratified 10-fold cross-validation on the whole dataset. With this strategy, we ensure that we have data from all types of devices in both training and test sets.

Considering multi-device RUMSI inference first, Table 4.5 and Figure 4.11a summarize the performance attained by a single ERT10 model, trained on multi-device data. Compared to per-device specialized models (cf. the matrix diagonal in Figure 4.8), there is a marginal degradation for the corresponding multi-device model, mainly observed for desktop, with an error increase close to 10%. Still, performance is consistent across the three device types, with an overall median absolute error of 327 ms, and a relative error of 14%. Overall, the generalization capabilities of the multi-device model outweigh the accuracy of the specialized models, making it a preferred choice for Web-QoE monitoring in operational deployments.

Considering multi-device QoE classification next, Figure 4.11b reports the performance obtained with a single ERT10 model, trained on multi-device data: again, we see a slight



(a) RUMSI-inference performance.



(b) QoE-classification performance.

Figure 4.11: RUMSI-inference and QoE-classification performance, the ERT10 model was trained on multi-device data.

performance degradation compared to the specialized desktop model (cf. Table 4.3), but with significant gains in terms of generalization to mobile devices (cf. Figure 4.9). The overall model accuracy is 82.2%, with recall and precision values for **excellent {e}**, **good {g}**, and **poor {p}** QoE of about $\{89\%, 73\%, 80\%\}$ and $\{86\%, 74\%, 82\%\}$, respectively.

Overall, our evaluation shows that a single multi-device model significantly improves generalization of the Web-QoE inference across different devices, with only slight underperformance as compared to specialized models applied to data coming from the corresponding device type. As such, multi-device models provide simple, more accurate, and more reliable monitoring capabilities in realistic Web-browsing scenarios.

4.5 Multi-Device, Flow-Level Models

In the last part of the study, we focus on improving the practical application of the proposed Web-QoE inference models. In particular, we explore the definition of new

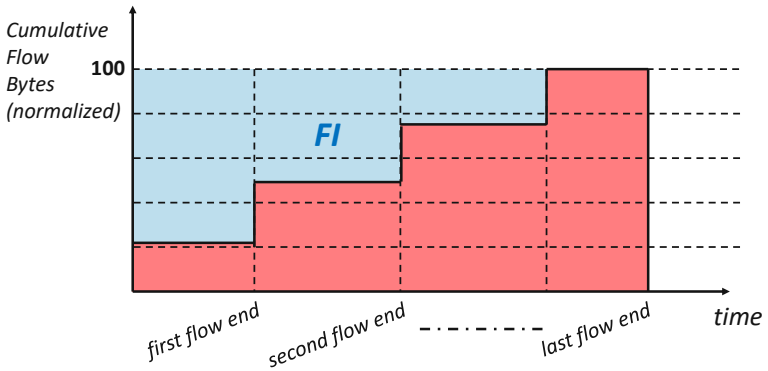


Figure 4.12: Flow-level features – FlowIndex computation.

input features at the flow level, with the intent of using fewer features and creating more lightweight models. We define a set of 21 flow-level features, using similar notions to the ones which guided the packet-level features. These include:

- The total number of flows (all, downlink, uplink).
- The min/mean/median/max flow duration in downlink.
- The min/mean/median/max flow size in downlink.
- The min/mean/median/max flow ByteIndex in downlink.
- The mean/median in-flow, average intra-packets time (MDT) in downlink.
- The mean/median/max flow throughput in downlink.
- The FlowIndex (FI).

The flow ByteIndex uses the standard definition of ByteIndex (BI) [30], but considering only the packets belonging to a specific flow.

The FI feature represents an extension to the BI, using flow size and flow ending time instead of packet size and time. Figure 4.12 depicts the basic notions behind the calculation of the FI. Both the FI and BI are integral metrics, similar to the definition of the SI.

Flow features are complemented by the 11 session-level features, previously defined in Section 4.2.3, for a total of 32 input features. In the context of this last analysis, we use the same Web sessions as before, replacing the packet-level features with the newly defined flow-level descriptors.

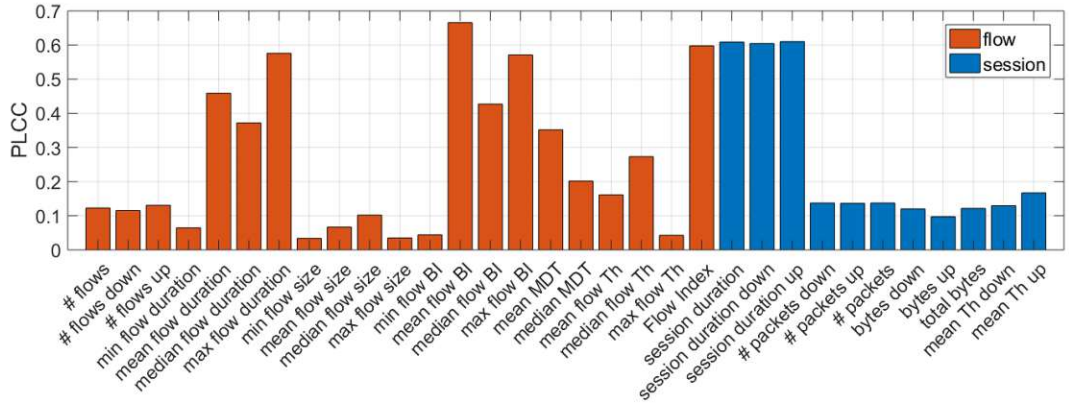


Figure 4.13: Flow-level features, ranked by PLCC.

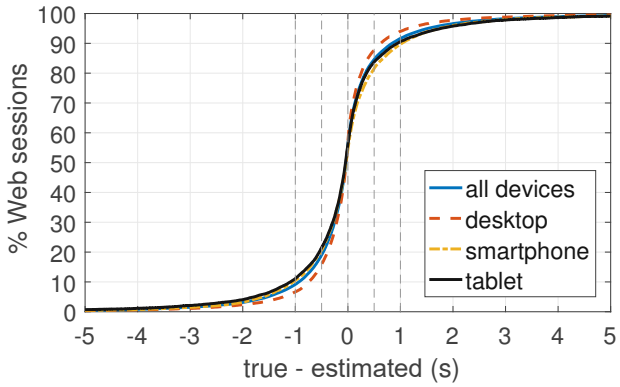


Figure 4.14: Multi-device RUMSI-inference performance – flow-level features.

For the sake of completeness, Figure 4.13 reports the linear correlation between these flow-level and session-level input features and the RUMSI metric. Features related to flow duration, flow BI, FI, and session duration are the ones showing the highest correlation to the RUMSI.

Figure 4.14 and Table 4.6 report and summarize the RUMSI-inference performance achieved by a multi-device model based on ERT10. Results are comparable, and even slightly better for some device types, than those obtained by using packet-level features (cf. Table 4.5), with the paramount advantage of using an order of magnitude fewer features, while being easier to compute and more scalable.

device	MAE-mAE (ms)	MRE-mRE (%)
desktop	628 – 309	35 – 18
smartphone	815 – 364	26 – 13
tablet	751 – 317	25 – 11
all	732 – 324	29 – 13

Table 4.6: Multi-device, flow-level RUMSI-inference performance.

4.6 Conclusions

We have tackled the problem of Web-QoE monitoring from the ISP perspective, relying on in-network, passive measurements. We built a large, multi-device, heterogeneous corpus of Web-QoE measurements for the most popular Websites in order to conduct empirical evaluations. They highlight that the proposed solution can infer the RUMSI as well as estimate Web-QoE ranges from in-network traffic measurements with high accuracy. Thanks to the heterogeneity of our data, we believe that our ML models have the potential to perform well in the wild. At the same time, we showed that the device type introduced a strong bias in the capabilities of Web-QoE inference models, causing models trained for single device types to badly generalize to other devices. We demonstrated that this lack of cross-device generalization can be solved by properly training on data coming from a multitude of devices. Our findings raise awareness of the fact that models for Web-QoE monitoring must be exposed to multi-device measurements to achieve good inference performance in real network deployments; this insight is highly relevant for ISPs, but is generally neglected in the literature. To the best of our knowledge, we are the first to unveil the strong impact that the device type has on such models. The definition of novel flow-level features which also yield highly accurate estimations is an additional key contribution of the study, as these kinds of lightweight statistics allow ISPs to create even more scalable models.

CHAPTER 5

(Adaptive) Detection of Network Attacks

Notice

Parts (text as well as figures, tables, and algorithms) of this chapter have already been published in the following publications:

[J4] *Adaptive and Reinforcement Learning Approaches for Online Network Monitoring and Analysis*
S. Wassermann, T. Cuvelier, P. Mulinka, P. Casas
IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 1832-1849, 2021
DOI: 10.1109/TNSM.2020.3037486
© IEEE 2021

[A1] *RAL – Reinforcement Active Learning for Network Traffic Monitoring and Analysis*
S. Wassermann, T. Cuvelier, P. Casas
SIGCOMM Posters, Demos, and Student Research Competition, August 2020
DOI: 10.1145/3405837.3411390

[C4] *ADAM & RAL: Adaptive Memory Learning and Reinforcement Active Learning for Network Monitoring*
S. Wassermann, T. Cuvelier, P. Mulinka, P. Casas
International Conference on Network and Service Management (CNSM), October 2019*
DOI: 10.23919/CNSM46954.2019.9012675

[W3] *RAL – Improving Stream-Based Active Learning by Reinforcement Learning*
S. Wassermann, T. Cuvelier, P. Casas
European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Workshop on Interactive Adaptive Learning, September 2019

[W1] *BIGMOMAL – Big Data Analytics for Mobile Malware Detection*
S. Wassermann, P. Casas

SIGCOMM Workshop on Traffic Measurements for Cybersecurity, August 2018
DOI: 10.1145/3229598.3229600

The MAWILab datasets [135] on which we evaluate RAL have been gathered, processed, and described by researchers of the CTU Czech Technical University in Prague (Section 5.3.5 *Data Description*). The work about BIGMOMAL has been carried out exclusively by myself. RAL has been conceived, implemented, and described with my co-author Thibaut Cuvelier.

Today, more and more tasks are performed on smart devices, and they see an impressive popularity: while 472,000,000 smartphones were sold in 2011 [136], this number has more than doubled in 2014 [137]. Moreover, users spend most of their time browsing on their mobile phones, more than on any other device. In the United States and the United Kingdom, for instance, smartphone users spend 2 to 3 times more hours (87 hours per month in the US and 66 hours in the UK) on their mobile device than on desktop machines (34 hours and 29 hours, respectively) [138]. The impressive growth of mobile devices and usage has led to unprecedented increase in cellular traffic. Unfortunately, these devices increasingly become the target of malicious attacks that aim at stealing the users' private information and using it for bad-natured activities. Smart devices are not only an appealing target because of their popularity, but also because they incorporate a very large amount of sensitive information about the users, even more than a personal computer [4]. For instance, the included sensors can track the user's current location and physical activities, and the camera can take pictures and record videos, even without the users' awareness.

The first piece of malware on a smart device was detected before 2004 [4] and the number of malware detections and their diversity grew with the expansion of the smart-device market. In particular, popular mobile operating systems such as Symbian, iOS, and Android accelerated the development of mobile malware: while only about 500 malware infections were reported between 2004 and 2007, this number increased by 3,325% in 2011 for Android [139]. In 2016, Kaspersky Lab found approximately 8,500,000 malicious installation packages, 130,000 mobile banking Trojans, and 260,000 mobile ransomware Trojans [140]. They noted that a large part of malicious software is distributed via the official app stores. From 2017 to 2019, the number of malicious packages decreased, but unfortunately this number increased again significantly (by 2,100,000) in 2020 [1]. Figure 5.1 depicts the types of malware which have been seen the most in 2019 and 2020. 2016 also underlined that developers of malicious apps make use of the current trends: an app called *Guide for Pokémon Go* including a Trojan made its appearance on Google Play and was installed more than 500,000 times [141].

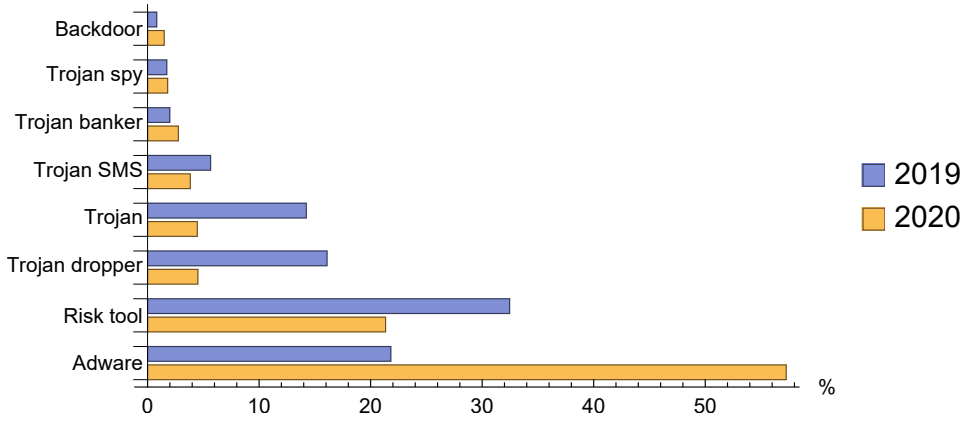


Figure 5.1: Most seen malware types in 2019 and 2020 for Android OS (data from Kaspersky's Mobile Malware Evolution 2020 report [1]).

Detecting mobile malware is a very challenging task and subject to a lot of research efforts. Unfortunately, Zhou et al. found that the best antivirus software merely detected 79.6% of malware [142], which is a rather disappointing result.

With the rise of IoT (Internet of Things) devices, the playground for cybercriminals becomes even larger. Indeed, this kind of device is used in many different domains such as the health-care, food, and energy sectors, and encompasses highly sensitive data. Due to the variety of devices and ways to connect to each other, we are facing a lack of standardization of protocols at the communication and data-audit level, which gives the threat actors more possibilities for attacks [143].

In this chapter, we first discuss the related work in the fields of mobile malware and active learning, a subdomain of ML which proves to be very promising for networking problems. We then introduce two frameworks that we apply in the context of cybersecurity:

1. BIGMOMAL, Big Data Analytics for Mobile Malware Detection. BIGMOMAL is a framework relying on supervised learning to identify running Android applications and to detect malware activity.

2. RAL, Reinforced stream-based Active Learning, a stream-based active-learning strategy coupled with reinforcement learning to reduce the amount of labeled data needed for learning.

For both BIGMOMAL and RAL, we present their functioning, the evaluations we conducted with them, and the achieved results. The last section concludes this chapter.

5.1 Related Work

Detecting mobile malware is a very challenging task and subject to significant research efforts. Zhou et al. made the rather disappointing observation that the best antivirus software merely detected 79.6% of malware [142]. It is worth noting that the wide range of touch commands (such as swiping and tapping on the display) adds another layer of complexity to the process of malware detection, as this makes the exploration of all execution paths almost impossible [144]. Nevertheless, the authors of Malton [145] developed a system which promises to be a non-invasive Android-malware-detection tool outperforming multiple other proposals.

According to [4], Android OS relies more on *platform protection* than on *market protection*. In other words, Android users can download applications even from unofficial markets, but the operating system attempts to protect the users from malicious behavior by, for instance, applying a permission-based system (i.e., an application can only perform the actions for which the permission has been granted) or letting applications run in a sandbox (i.e., an environment isolating the app). Non-negligible efforts have been spent to detect the overprivilege problem, where an application requests more permissions than it needs to work properly. Projects such as Stowaway [146], PScout [147], and DroidRisk [148] try to identify applications incorporating this kind of security threat. The work presented in [149] uses permission-based heuristics to determine whether an application is malicious or not.

In [150, 144], Arshad et al. and Tam et al. provide extensive surveys about malware detection and protection on Android. They present two approaches for detecting malware on Android phones: static and dynamic ones. While the static techniques are mainly based on bytecode analysis, without executing the app, dynamic methods analyze applications during their execution. For example, static techniques involve signature and permission analyses, whereas dynamic ones use anomaly-detection approaches and emulation. The aforementioned works [146, 147, 148, 149] fall into the category of static-analysis systems. TaintDroid [151] and Maline [152] are two examples of systems making use of dynamic techniques. TaintDroid relies on captured network data to analyze Android applications, while Maline is a malware-detection tool based on machine-learning techniques to classify system calls. Another tool relying on a dynamic approach has been developed by Afonso et al. [153]: it investigates the system calls and API uses of an application to feed extracted features to an ML algorithm. With MalDozer [154], Karbab et al. use more complex algorithms, in particular deep learning, to detect malware on Android devices.

To help the cybersecurity research community, several researchers make their mobile-malware datasets publicly available. A good example is the SherLock dataset [155], a massive time-series dataset containing more than 600,000,000,000 samples. More precisely, they analyzed the activities of Android applications in a very fine-grained fashion and simulated malicious behaviors on the monitored smartphones. Other examples are the Device Analyzer dataset [156], a very large dataset including data collected from

nearly 900 different devices showing the behavioral diversity among users, the LiveLab Project [157], a dataset investigating the usage of iPhone 3GS smartphones, and the Android Malware Genome Project [142], a dataset encompassing more than 1,200 different Android malware samples from nearly 50 distinct malware families.

Relying on ML to build novel malware-detection systems is on the rise, due to its promising results in a wide range of domains and its first applications in the field of cybersecurity. Network security is a very dynamic field, in the sense that security threats evolve very quickly and users, as well as security systems, need to be watched out for new threats constantly. Even though ML is getting used more and more often, most techniques still rely on offline-learning approaches, i.e. models are trained once and then applied to the incoming measurements. However, this is far from ideal when we are facing highly dynamic environments, where the underlying properties of the inference target evolve often and previous knowledge rapidly becomes obsolete. Another challenge we need to consider in network security is the issue of data sparsity: even though the total amount of data is overwhelming, not all attacks are equally covered, and a lot of the captured traffic is (thankfully) benign.

To tackle those problems, we developed an active-learning framework coupled with reinforcement learning.

Many research efforts have already been undertaken in the field of active learning. For example, [158, 159] present three simple approaches for this learning paradigm. Their Randomized Variable Uncertainty approach tackles the problem of stream-based active learning using the model's inference uncertainty to decide whether to query. They try to adapt to concept drifts by randomizing the certainty threshold used for querying decisions. [160] develops an active-learning algorithm with two different classifiers: one "reactive" and one "stable". The stable classifier is trained on all available labeled instances, while the reactive one learns on a window of recent instances. In [161], the authors present an active-learning technique based on clustering and estimation uncertainty. [162] conceives an approach relying on a modification of the Naive-Bayes classifier to update the different learners through the queried samples. In particular, the author uses one-versus-one classifiers to tackle multi-class problems and update the weights of the different classifiers by comparing their estimations to the ground truth. Krawczyk's technique behaves similarly to ours. However, the major difference is that he is using information about the classifiers' inference certainty (without considering the corresponding weights) in order to adapt the minimum threshold to query the oracle, while we rely on the usefulness of the decisions taken by RAL to tune the system according to the data stream.

Extending active learning through reinforcement learning is currently a very active research area. Active learning alone can easily converge on a policy of actions that have worked well in the past, but are sub-optimal. Reinforcement learning helps to improve the exploration-exploitation trade-off by letting the learner take risks with an uncertain outcome. However, most proposals do not consider the stream scenario, and operate on the basis of pool-based approaches. In [163, 164], authors rely on the multi-armed

bandit paradigm. [164] develops ALBL, which uses a modified version of EXP4 [165], a weight-updating rule, to attribute adaptive weights to different learners based on rewards; the learner to use is then determined through these weights; the chosen learner selects the samples in the pool to hand to the oracle based on its uncertainty measure. The approach described in [163] is similar to the one in [164], except for the reward-computation scheme. Some other papers in the field of pool-based active learning are [166, 167]. The algorithm presented in [168, 169] relies on the same principles as the system we are proposing, but tackles a different problem: Song's goal is to introduce an active-learning component into a contextual-bandit problem, while we are aiming at solving an active-learning problem by using contextual bandits.

Other recent papers dealing with active learning and reinforcement learning include [170, 171, 172, 173]. However, most of them consider only one of the perspectives addressed by RAL, namely the enhancement of pool-based active learning through reinforcement learning [171, 173, 172], or the application of active learning to the streaming setup [170]. Combining active learning with reinforcement learning in a streaming, adaptive learning context is the most important contribution of RAL, a very timely yet vaguely addressed problem in the literature. [174, 175] use the idea of *learning to active learn*, i.e. data-driven active learning. [175] suggests this view on pool-based active learning: the querying decision for a sample is based on an estimation of the accuracy improvement. [174] uses reinforcement learning in stream-based active one-shot learning, but this work is different from RAL on multiple aspects:

1. It tackles a different learning task, as it aims at detecting new classes instead of improving overall classification accuracy for previously known classes in the training phase.
2. Their scheme relies on reinforcement learning only during the training phase and not once deployed, while RAL continuously adapts its querying policy during the whole incoming stream.
3. The system heavily relies on deep recurrent networks, too cumbersome to use in real-time resource-constrained scenarios, unlike RAL, which is model-independent.

5.2 In-Device Mobile Malware Detection with BIGMOMAL

Given the relevance and sensitivity of mobile cybersecurity, we study the problem of malware and running-application detection in Android smartphones relying on supervised-machine-learning models. We use a large, publicly available dataset for smartphone data analysis, namely the SherLock data collection[1], to build BIGMOMAL, a framework to detect malware activity through supervised machine learning. We evaluate three different aspects:

[1] http://bigdata.ise.bgu.ac.il/sherlock/

1. Overall model performance.
2. Generalization of the learned models across different users.
3. Detection-accuracy drift along time.

Initial results suggest that decision trees are capable of identifying malware activity with high accuracy and extremely low computational time.

Our contributions to the mobile-security domain are as follows:

- **Running-application identification:** we identify running Android applications for each user through lightweight ML models with resource-level features as input. We also investigate the impact of time on this identification, i.e. how our models behave when they are trained on a different time period than the one on which they are tested (for example, trained on data at the beginning of the month, and tested at the end of it).
- **Malware detection:** we use the same input features as for the running-app identification to detect malicious applications. Besides a temporal analysis, we also evaluate ML models when mixing data from multiple users.

In Section 5.2.1, we describe the Sherlock dataset and its characteristics. In Section 5.2.2, we present how we use BIGMOMAL to identify running applications on Android devices. In Section 5.2.3, we rely on BIGMOMAL to detect potential Android malware.

5.2.1 The SherLock Dataset

In the context of this work, we rely on the SherLock dataset, published by the BGU Cyber Security Research Center [155]. The SherLock dataset has been collected from January 2015 until December 2017 and includes very valuable information about the usage of Android smartphones. The collection was done during a long-term field trial on 50 smartphones used as primary device for different participants. More precisely, Mirsky et al. handed a Samsung Galaxy S5 to 50 clients. On these smartphones, the authors installed two applications, namely SherLock and Moriarty: while SherLock is responsible for collecting data about the smartphone (SMS, call logs, etc.), Moriarty periodically performs malicious activities, simulating the behavior of malware found in the wild. SherLock monitors two kinds of sensors:

1. PULL sensors, analyzed regularly (to collect, for instance, information about the installed and currently running applications, and the values of accelerometers).
2. PUSH sensors, triggered when an event occurs (for example, receiving / issuing a call or a text message).

The malicious actions of Moriarty are diverse and encompass activities such as contact, photo and SMS theft, phishing, and ransomware: its payload is updated every few weeks. Data collection is done at very high sample rate (as frequently as once every 5 seconds). The complete labeled dataset contains more than 600,000,000,000 data records, with a total of more than 4 TB of data. It is worth noting that the SherLock team cares a lot about the users' privacy: network identifiers such as cell-tower IDs, WiFi SSIDs, and MAC addresses have been hashed; geographical locations have been anonymized through clustering techniques.

In this study, our evaluation focuses on the data collected during the second quarter of 2016. From the complete dataset, we keep two specific feature categories:

1. All those features related to the network traffic generated by the apps.
2. All those features corresponding to the footprint of the app on the CPU and internal running processes (e.g., statistics on CPU and memory usage, Linux-level process information).

The rationale is that some malware activity would be more visible at the network-traffic level, whereas some others would be better identified at the local-process level. The full list of used features can be found in Appendix D.

Given the size of the data we study, we rely on big-data platforms to conduct the study. In particular, we use the Big-DAMA framework [176] to process the dataset, build machine-learning models, and evaluate their performance. In a nutshell, Big-DAMA uses Apache Spark Streaming for stream-based analysis, Spark for batch analysis, and Apache Cassandra for query and storage. Within the Big-DAMA platform, we have conceived different algorithms for network security and anomaly detection using supervised- and unsupervised-machine-learning models.

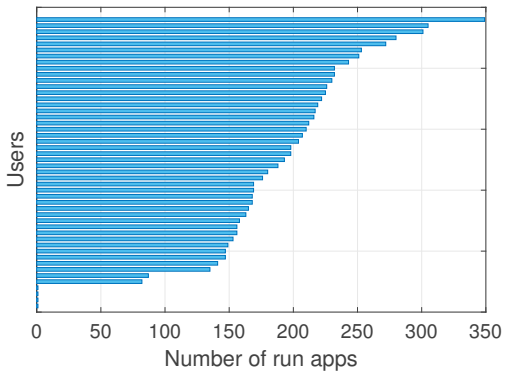


Figure 5.2: Number of different applications launched during Q2 2016 by each user.

We start by doing a brief characterization of the considered dataset snippet. During these three months, we found more than 600,000,000 records from 48 different users.

Rank	Application name	# samples
#1	SherLock	88,770,706
#2	Google Play Services	40,618,023
#3	Chrome	39,826,584
#4	SmartcardService	37,303,817
#5	Unified Deamon	37,151,289
#6	SecurityManagerService	37,139,510
#7	WhatsApp	36,427,517
#8	S Finder	35,370,165
#9	Samsung Push Service	34,685,712
#10	Context Service	31,030,179

Table 5.1: Top ten popular apps in Q2 2016.

Figure 5.2 depicts the number of different Android applications launched by each user. We can easily see that not all the users use their smartphone in the same way: while nearly 50% of them launched more than 200 apps, there are also clients who used less than 10 different applications. For the latter, we suppose that they did not use the Samsung smartphone on a daily basis, but only periodically for the SherLock campaign. Next, we check which applications were the most popular among our users. We consider an app as popular if the dataset contains a large number of data points for it. Table 5.1 shows the ten most popular apps for Q2 2016. For Moriarty, we observe 7,330,805 data points. We note that SherLock was the app yielding by far the most samples (12 times more), which comes as no surprise because it is the application responsible for the information gathering. Through this table, we can also point out that a non-negligible part of the popular applications are Android services which are automatically launched in the background by the Android OS and not by the users. For instance, SmartcardService ensures that applications can communicate with the SIM card and Google Play services allows the installed applications to access the newest features published by Google. In Table 5.2, we summarize the ten most popular apps excluding internal Android services and the SherLock application. According to our findings, WhatsApp and Hangouts seem to have been the favorite social applications of our users from April to June 2016.

To better understand the behavior of Moriarty as compared to benign applications, we compare WhatsApp – the most popular social app in the dataset – against Moriarty, the malware-emulation app. In Figure 5.3, we plot several characteristics of both of them. In particular, we focus on the average CPU utilization per hour, the average number of used threads per hour, and the sent / received traffic on a hourly basis. We observe that Moriarty used much more CPU resources than WhatsApp: while WhatsApp used most of the time only 5% of the CPU on the monitored smartphones, Moriarty mostly

Rank	Application name	# samples
#1	Chrome	60,906,775
#2	Google App	39,826,584
#3	WhatsApp	36,427,517
#4	S Finder	35,370,165
#5	S Health	24,584,561
#6	Hangouts	19,787,131
#7	Samsung text-to-speech engine	18,538,122
#8	Geo News	17,132,911
#9	Peel Smart Remote	15,729,303
#10	Contacts	14,995,777

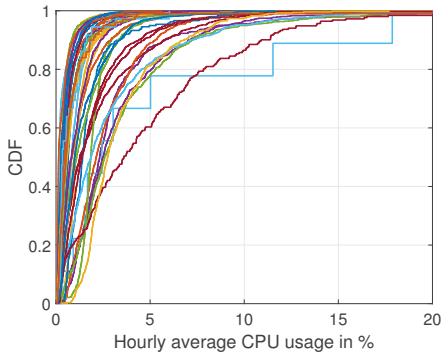
Table 5.2: Top ten popular apps in Q2 2016 (Android services and SherLock excluded).

needed at least 20%, which is a significant difference. However, WhatsApp seems to need more threads to run properly compared to Moriarty. When it comes to network traffic, we can see that WhatsApp sends more data than Moriarty. We observe a similar behavior for the received traffic, even though the difference is not as striking. While this is expected, as the users of WhatsApp constantly exchange messages and multimedia files, this simple comparison makes the point in the statistical differences in some of the aforementioned characteristics, which shall form the basis for the input features to the machine-learning models detecting malware activity.

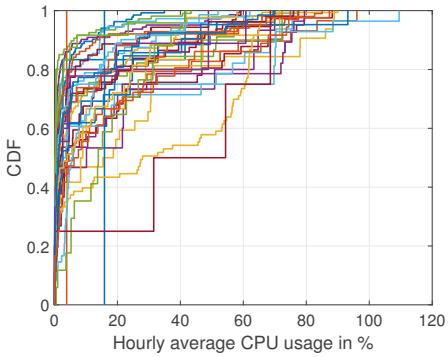
5.2.2 Identifying Running Applications

Before setting the focus on the malware detection, we investigate the possibility of automatically identifying the different running Android applications from the set of input features available in the dataset. The goal is to differentiate among different running apps without gathering the process name, so as to better understand if it would be possible to later on identify malware activity using the same approach.

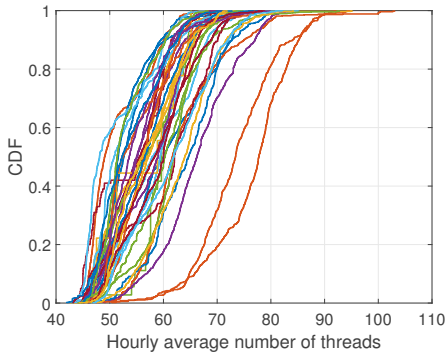
We consider as input a set of 45 SherLock-monitored smartphone features (see Appendix D), reflecting the behavior of the different apps, and build supervised-machine-learning models to automatically classify them. We particularly focus on the usage of decision trees, based on their a-priory excellent performance in similar network-measurement analysis problems, as we saw in Chapters 3 and 4 and in [177]. We proceed as follows: for each user, we randomly choose 20% of the application samples as training set and the remaining 80% for testing. We obtain almost perfect results for the entire set of Q2 users: for all of them, our model correctly identified the running applications in about 99% of the cases.



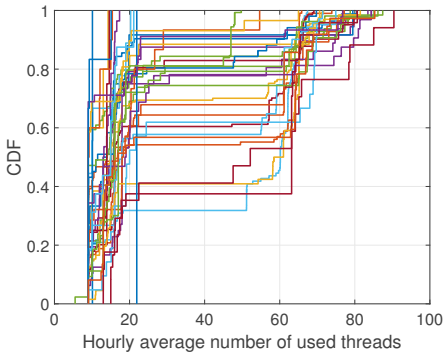
(a) Avg. CPU utilization / hour for WhatsApp.



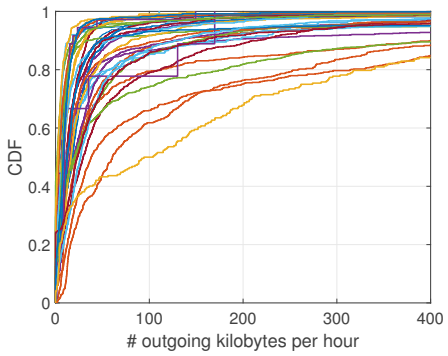
(b) Avg. CPU utilization / hour for Moriarty.



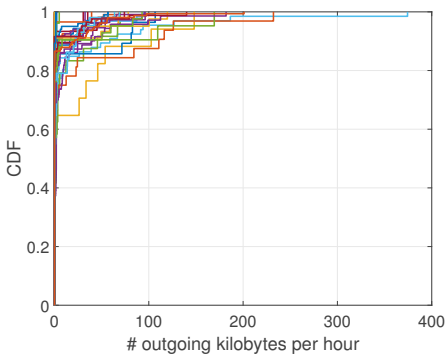
(c) Avg. # used threads / hour for WhatsApp.



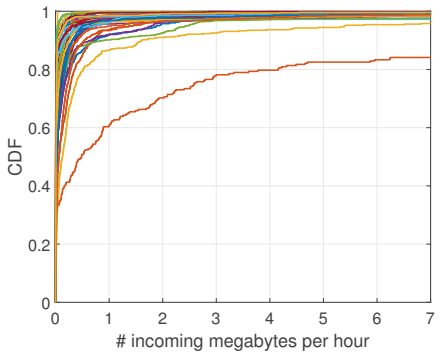
(d) Avg. # used threads / hour for Moriarty.



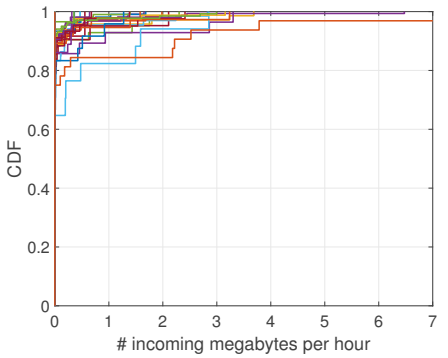
(e) # sent kilobytes / hour by WhatsApp.



(f) # sent kilobytes / hour by Moriarty.



(g) # received megabytes / hour by WhatsApp.



(h) # received megabytes / hour by Moriarty.

Figure 5.3: Characterization of the WhatsApp and Moriarty Android applications. Each CDF curve on a plot corresponds to the cumulative distribution for one user and the sampling frequency is once per hour.

Top 10 features	Application identification	Malware detection
#1 feature	virtual memory size	shared dirty pages used by Dalvik heap
#2 feature	number of threads	number of minor page faults
#3 feature	CPU utilization	time process scheduled in user mode
#4 feature	process priority level (foreground, background, service, sleeping, etc.)	proportional set size for Dalvik heap
#5 feature	ordering within a particular priority category	time process-children scheduled in user mode
#6 feature	process life time	virtual memory size
#7 feature	time process scheduled in user mode	time process scheduled in kernel mode
#8 feature	time process-children scheduled in user mode	process ID
#9 feature	number of minor page faults	process life time
#10 feature	number of private dirty pages used by everything else than the native heap	number of private dirty pages used by the native heap

Table 5.3: Most relevant features for identifying running Android applications and detecting malware.

Feature Selection

Next, we investigate which features are the most discriminative ones when it comes to the identification of applications. We rely on an embedded method within a decision tree. The ten most important features are reported in Table 5.3.

We note that the most discriminative features are related to the CPU, threads, and memory. Moreover, the information related to the importance of the Android application – i.e., whether it is running as a service, in the background, or in the foreground – helps for the identification. Overall, our results show that low-level information is very useful for this kind of inference task. Interestingly, the feature-importance scores show that we do not have any feature with an importance close to 1; instead, the highest is 0.32. This shows that the ML technique can combine these features in a very smart way for an accurate inference.

As an additional analysis, we evaluate the application-identification power of a decision tree for one user when relying only on the five and ten most important features. For

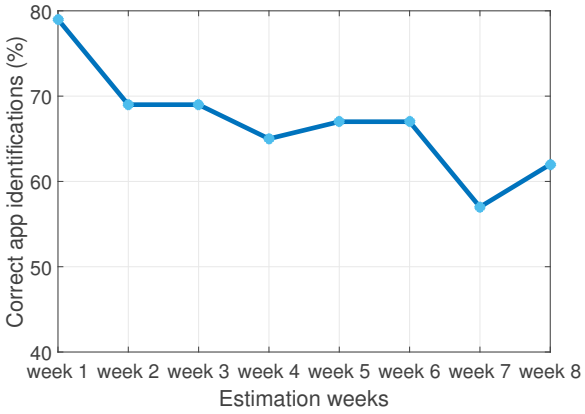


Figure 5.4: App-identification accuracy when training and testing on a weekly basis.

this, we split the dataset as previously, i.e. 20% of the data is considered as the training set and the remaining 80% as test set. Our results reveal that the decision-tree model still works extremely well even when using only 5 or 10 features instead of 45: in both cases, the identification was correct for about 99% of the testing records. It thus seems that only a small amount of data is required to obtain a high estimation accuracy, which is a paramount advantage in case the app identification needs to be done in an online context. Indeed, when working with a restricted set of features, classifiers can work more efficiently and provide an answer quickly. Furthermore, gathering only little data implies that we do not have to overload the smartphone for data collection – only a few system calls are necessary.

Temporal App Identification

As a last step for the app-identification study, we are interested in the quality of the application identification on a temporal basis. More precisely, we are curious about the accuracy of the inference model in case the training phase takes place in a different time span. In particular, for each user, we train a decision tree on data collected during the first week and try to identify the applications run during the following weeks.

Figure 5.4 depicts the results for one user. It reveals that the overall identification accuracy decreases with respect to our previous evaluation, suggesting that restricting the training set to the samples collected during the first week does not capture as well the users' smartphone habits as when the training set is randomly sampled. As a concrete example, a user could be traveling during the seventh week and send a lot of photos to her friends via WhatsApp, which would significantly alter the network-related features of this application. Furthermore, the plot shows that the accuracy of our model decreases over time: while almost 80% of the applications were correctly identified in the first estimation week, this was the case for only about 62% in the eighth one. The outcome of this analysis for the other users is very similar.

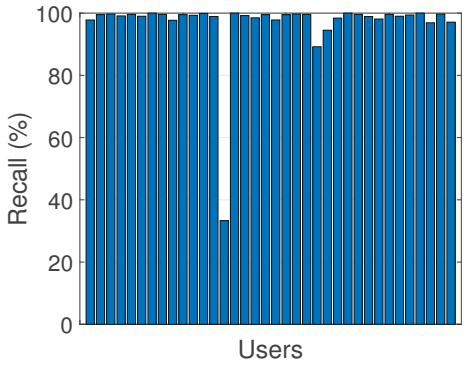


Figure 5.5: Recall obtained while detecting Moriarty for each Q2 user.

	Inferred negative	**Inferred positive**
Ground-truth negative	14,275,309	117
Ground-truth positive	155	35,076

Table 5.4: Confusion matrix obtained while detecting Moriarty for one Q2 user.

5.2.3 Malware Detection

Our next goal using BIGMOMAL is to detect whether an application is malicious or not, still working with the data collected during the second quarter of 2016. As presented in Section 5.2.1, Moriarty simulates bad-natured behavior and is thus the target of the detection model. We are now trying to answer the following question: *"Is the running application Moriarty or not?"*, which boils down to a binary-classification problem.

We use the same feature set and ML model as for the application-identification task. However, as the execution of the Moriarty application can be considered as a rare event (the dataset contains millions of non-Moriarty samples versus only a few thousands of Moriarty records for each user), we randomly choose 40% of samples for the test set compared to 20% in the context of the application identification.

Figure 5.5 depicts the obtained recall for the Moriarty detection per user. A high recall value reveals that the number of missed malware instances (the false negatives) is low. This is a very important property in our context, as false negatives (i.e., considering an application as benign although it is malicious) are a considerable security threat for the user. Moreover, we achieve an extremely low false positive rate (below 1% for each user), indicating that we almost never classify a benign application as malicious. For the sake of illustration, Table 5.4 shows the confusion matrix obtained for one of our users.

Feature Selection

Similar to the identification of applications, we are interested in the most important features for detecting malicious apps. The ten most relevant features (determined using the same technique as in Section 5.2.2) are summarized in Table 5.3. We note that, compared to the application-identification task, a lot of these features are related to the memory (five out of ten). Once again, low-level metrics are very useful indicators. Similar to the app identification, we do not observe any importance score close to 1. As a final step, and similarly to Section 5.2.2, we evaluate our decision-tree model for this detection problem with the top five and top ten features. The results are excellent even with these reduced feature sets. Indeed, in both cases, the recall is over 99%.

Temporal Malware Detection

Next, we carry out the temporal analysis for the Moriarty detection. We proceed in the same way as for our previous task, i.e., we train for each user the detection model on data gathered during the first week of the second quarter and test it in the remaining weeks. Given the major imbalance in classes between benign and Moriarty instances, we resort to SMOTE [111] to oversample the Moriarty class to achieve a 50-50 ratio between the majority and minority classes.

Figure 5.6 depicts the recall obtained per week for one user. As we can easily observe, the results are very poor and detection performance rapidly degrades after the 3rd week. Two issues could be the cause of this performance: firstly, the extremely low number of gathered Moriarty samples in the training week, and secondly, a high level of dynamism in the data; as stated in Section 5.2.1, the Moriarty application periodically changes its behavior, which makes its detection very challenging when the training set does not include samples of all the patterns. Indeed, during the second quarter of 2016, Moriarty changed twice its malicious behavior to emulate several kinds of malware: it started with SMS- / bank-theft malware activity, moved on to phishing, and finished with simulating adware [155]. For instance, displaying ads intuitively requires more network traffic and CPU usage than SMS/bank theft and phishing. This prevents the decision tree from accurately grasping the behavior of the malicious Android application. Again, the results for the other users are highly similar. We tested several other models for this specific task, without any notable detection improvements.

Malware Detection Across Multiple Users

Finally, we evaluate our detection model on data collected from multiple users. More precisely, we train our decision tree on data gathered from three Q2 users and test it on data belonging to three other ones.

The resulting confusion matrix is shown in Table 5.5. Results are again quite disappointing, suggesting that the particularities of the ways users use their smartphone play a more important role than the set of features used as input by the model. Indeed, each

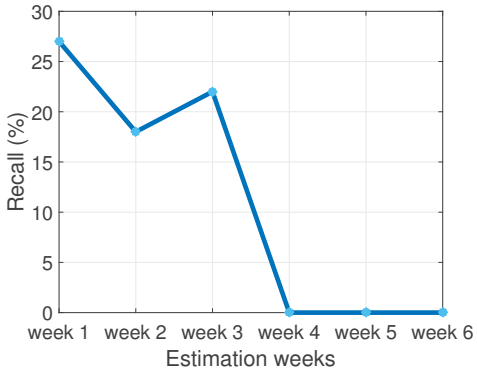


Figure 5.6: Recall obtained while detecting Moriarty on a weekly basis for one Q2 user.

	Inferred negative	**Inferred positive**
Ground-truth negative	79,775,923	16,784
Ground-truth positive	35,401	3,472

Table 5.5: Confusion matrix obtained while detecting Moriarty for several Q2 users when training on data instances from other clients.

and every user uses her smartphone differently and it is thus very challenging to build a model from multiple users that can be accurately applied to other ones.

5.3 Reinforcement Learning for in-Network Attack Detection with RAL

One of the main challenges associated with supervised learning under dynamic scenarios is that of periodically getting access to labels of fresh, previously unseen samples. Labeling new data is usually an expensive and cumbersome process, and not all data points are equally valuable. A good example is again the domain of cybersecurity, where manually analyzing and labeling potentially malicious samples is both time- and effort-consuming. Another challenge we are facing in security, and in network monitoring in general, is the one of fast incoming and heterogeneous data. This data usually comes in the form of high-speed streams, which need to be rapidly and continuously processed. In this context, detecting and adapting to strong variations in the underlying statistical properties of the modeled data makes data-stream analysis a challenging task.

Active learning aims at labeling only the most informative samples to reduce cost. In this paradigm, a learner can choose from which new data points it wants to learn, and can obtain the ground truth by asking an oracle for the corresponding labels.

In this section, we introduce our contribution to the active-learning domain: RAL – Reinforced stream-based Active Learning –, a new active-learning approach, coupling

stream-based active learning with reinforcement-learning concepts. In particular, RAL's decision-making process is guided by rewards and penalties. To the best of our knowledge, combining stream-based active learning with reinforcement learning has not yet been explored in the literature. Additionally, we release RAL on GitHub as an open-source Python package to the machine-learning community[2].

In Sections 5.3.1 and 5.3.2, we present the general functioning of RAL. In Sections 5.3.3 and 5.3.4, we detail two different versions of RAL: the one using a committee of models and the one based on a single ML model. In Section 5.3.5, we benchmark RAL against state-of-the-art algorithms on two network-security datasets, and show that it outperforms the considered techniques.

Even though we evaluate RAL only on cybersecurity datasets in this thesis, we designed it to work in all situations where data sparsity is an issue. For instance, we benchmark our system on forest covertype data in [J4, W3].

5.3.1 Overview of RAL

In this subsection, we give a broad overview of RAL – Reinforced stream-based Active Learning – and its two building blocks. The overall idea is summarized in Figure 5.7.

RAL can be used within systems that have an incoming stream of data to label, one sample at a time; each data point is referred to as x and the corresponding ground-truth label y is not cheaply available through an oracle. The designers of these systems want to continuously improve their performance (typically, measured as a misclassification rate) while minimizing the number of useless sample labelings. The role of RAL is to decide, for each sample, whether it would be beneficial for the global system to have access to its label through the oracle.

To accomplish this goal, RAL is composed of two essential building blocks:

1. **A set of experts.** Each of them gives its own independent opinion on each incoming sample, indicating whether the corresponding label should be obtained.
2. **A controller that takes the final query decision based on the opinion of each expert.** This module is based on reinforcement learning and gives more importance to experts that have a good performance history (i.e. following their advice mostly lead to labeling samples that were useful).

5.3.2 Diving into RAL

In this subsection, we detail the design of RAL and its two main building blocks. RAL relies on inference uncertainty, a commonly used metric in active learning, and reinforcement-learning principles, using rewards and bandits, which are among the most studied algorithms.

[2] https://github.com/SAWassermann/RAL

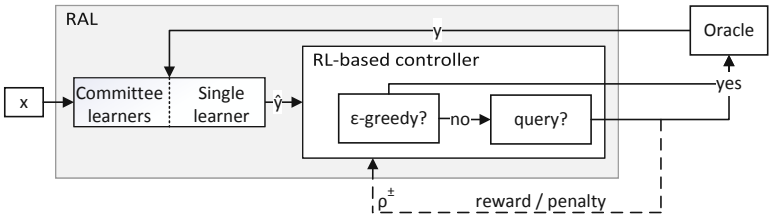


Figure 5.7: Overall idea of the RAL system.

Learners as Experts

In RAL's committee, each ML model is an expert, taking decisions based on its own certainty. The calculation of the inference certainty is independent of the way the classifier is trained. The certainty of the model is defined as the highest posterior classification probability among all possible labels for sample x. More formally, it is equal to $\max_{\hat{y}} P(\hat{y}|x)$ with $\hat{y}$ being one of all the possible labels for x. The uncertainty is thus defined as $1 - \max_{\hat{y}} P(\hat{y}|x)$.

The underlying assumption for designing RAL is based on the rationale that the expert's (un)certainty is an appropriate proxy for assessing the usefulness of a data point. Indeed, if the learner is uncertain about its estimation, this sample likely represents a region which has not been explored enough. Adding that sample with its true label to the training set would improve the overall inference accuracy; the alternative is that the uncertainty is due to noise or to concept drift, these two points being especially challenging in a streaming setting.

Reinforcement-Learning-Based Controller

The implemented controller is mainly driven by rewards. The intuition behind the different reward values is that we attribute a positive reward in case our system behaves as expected, and a negative one otherwise, to penalize it. RAL obtains rewards/penalties only when it is asking for ground truth. In a nutshell, it earns a positive reward ρ^+ in case it queries the oracle and would have inferred the wrong label otherwise (the system made the right decision to ask for the ground truth) and a penalty ρ^- (i.e. a negative reward) when it asks the oracle even though the underlying classification model would have estimated the correct label (querying was unnecessary). The rationale for using reinforcement learning is that RAL learns not only based on the queried samples themselves, but also from the usefulness of its decisions. The objective function to maximize is the total reward $\sum_{n=1}^{T} r_n$, where r_n is the nth reward (either ρ^+ or ρ^-) obtained by RAL.

In RAL, we use the reward mechanism to tune the sample-informativeness heuristic in order to better guide the future query decisions, which will be explained in the next

subsection. In case RAL's committee is composed of more than one expert, rewards and penalties also influence the weight of each expert.

As a final element, we implement an ε-greedy policy in our controller (also inspired by the bandit literature [178]) for the sake of data-space exploration: with probability ε, the system queries the oracle, even if the committee agreed not to query; we call this the ε-*scenario*. This ensures that we have a good chance of detecting potential concept drifts: without this policy, the system could end up being too confident about its estimations (and thus never ask the oracle again) even though its estimations are erroneous.

5.3.3 Learning with a Committee of Learners

In this subection, we present RAL coupled with a committee of learners. The algorithm behind this version of RAL is summarized in Algorithm 5.1. We rely on a set of experts (i.e. different ML models), referred to as a *committee*. Each expert gives its advice for the sample to consider: should the system ask the oracle for feedback or is the expert confident enough about its inference? To assess a model's estimation certainty, we rely on a certainty threshold θ:

- If the model's certainty is below θ, the model is too uncertain about the inference to make and thus it advises that RAL asks the ground truth.
- Otherwise, the model has at least a certainty of θ and querying the label seems to be unnecessary.

The query decision of the committee takes into account the opinions of the experts, but also their decision power: if the weighted majority of the experts votes for not querying, RAL will rely on the label inference provided by the committee, used in the form of a voting classifier. The decision power of each expert gets updated such that the experts which agree with the entire committee are obtaining more power in case that particular decision is rewarding, i.e. informative (otherwise, these experts get penalized). These weights are updated through the EXP4 rule [165], with a learning rate η; this algorithm is typically used for contextual bandits with a set of experts. RAL does not update the decision powers of the different learners in the ε-scenario: the committee did not take the querying decision and thus the weights of the models should not be impacted by this querying action.

The computation of the reward is carried out every time the committee decided to query (i.e. not in the ε-scenario). RAL therefore gets rewarded with ρ^+ when it queried the oracle and asking was rewarding (i.e. the voting classifier would have inferred the wrong label). Conversely, RAL is penalized with ρ^- if the system used the oracle because the committee decided to do so, even though the underlying classifier would have inferred the correct class.

Algorithm 5.1 RAL algorithm.

procedure RAL(x, E, α, θ, ε, η, $\rho^{\pm}$)
    x: sample to consider
    E: set of learners, members of the committee
    α: vector of decision powers of learners in E
    θ: certainty/querying threshold
    ε: threshold for ε-greedy
    η: learning rate for updating decision powers and the querying threshold
    $\rho^{\pm}$: reward and penalty
    decisions $\leftarrow$ {} ▷ will contain decisions of learners
    for e $\in$ E **do**
        decisions[e] $\leftarrow$ e.askCertainty(x) $< \theta$ ▷ decisions[e] $\in$ {0, 1}
    end for
    committeeDecision $\leftarrow$ round($\sum_{e \in E} \alpha[e] \cdot$ decisions[e])
    $p \leftarrow \mathcal{U}_{[0,1]}$ ▷ random number drawn from a uniform distribution
    if $p < \varepsilon$ **or** committeeDecision $= 1$ **then** ▷ ε-scenario or not?
        $y \leftarrow$ acquireLabel(x)
    end if
    if committeeDecision $= 1$ **then**
        $r \leftarrow$ getReward(x, y, $\rho^{\pm}$)
        $\alpha \leftarrow$ updateDecisionPowers(r, E, decisions, committeeDecision, α, η)
        $\theta \leftarrow \min\left\{\theta \cdot \left(1 + \eta \cdot \left(1 - 2^{\frac{r}{\rho^-}}\right)\right),\quad 1\right\}$
    end if
end procedure

function getReward(x, y, $\rho^{\pm}$)
    return (ρ^- **if** $\hat{y}(x) = y$ **else** ρ^+) ▷ $\hat{y}(x)$ is the predicted class for x
end function

function updateDecisionPowers(r, E, decisions, committeeDecision, α, η)
    for e $\in$ E **do**
        if decisions[e] $=$ committeeDecision **then**
            $\alpha[e] \leftarrow \alpha[e] \cdot \exp(\eta \cdot r)$ ▷ EXP4
        end if
    end for
    $\alpha \leftarrow \alpha / \sum_{e \in E} \alpha[e]$ ▷ normalize each value of α
    return α
end function

As an additional step, to ensure that RAL adapts as best as possible to the data stream, we do not only tune the weights of the committee members based on rewards, but also the certainty threshold θ, denoted in the remainder of this subsection as θ_n to stress that

it is influenced by the $n-1$ samples observed so far. Again, as for the decision powers, θ_n is not updated in the ε-scenario.

The update rule of θ_n we implemented for our tool is written as follows:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \cdot \left(1+\eta \cdot \left(1-2^{\frac{r_n}{\rho^-}}\right)\right), \quad 1\right\} \tag{5.1}$$

Design of Update Rule

In this subsection, we detail the reasoning behind our choice of the update policy. Most active-learning algorithms only rely on a static or random certainty threshold (like Randomized Variable Uncertainty [158, 159]), while reinforcement learning provides the opportunity to have a threshold that adapts to the system's current state.

We are looking for a rule of this form:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \cdot (1+f(r_n)), \quad 1\right\}, \tag{5.2}$$

where $f(r_n) = 1 - \exp(a \cdot r_n)$ with $a < 0$ is a function indicating how the threshold should change: the threshold should increase when f takes positive values, decrease when f is negative.

The design goals of this update policy are that the threshold increases slightly when the reward is positive, conversely when the reward is negative. More formally, our update policy should satisfy the following properties:

1. θ_n **should decrease *rapidly* in case r_n is negative**, as this indicates that the system queries too often and thus is performing poorly. Therefore, θ_n should be adapted fast to improve its performance.
2. θ_n **should *slightly* increase when r_n is positive**, so that the system does not keep decreasing the threshold. The model was right to ask for more samples, and thus the threshold should be increased. Nevertheless, the system is doing well: the threshold should not be too reactive to the queries.
3. f **must have two extrema**: a minimum at $\rho^- < 0$ and a maximum at $\rho^+ > 0$, while being monotonic in between.
4. θ_n **represents a probability**. $\theta_n = 0$ is not acceptable due to the product form of the update policy, thus the values of θ_n must be in the interval $(0, 1]$.
5. $f(r_n)$ **must be in the interval** $(-1, 1]$ to ensure that θ_n takes values corresponding to a probability. We exclude -1 from the allowed range of values to avoid that θ_n drops to 0.

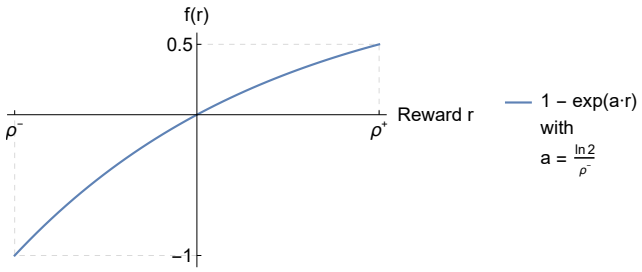


Figure 5.8: Function based on the reward used in our update rule of RAL's certainty threshold.

Properties 1 and 2 lead us to choose the family of functions $f_a : r \mapsto 1 - \exp(a \cdot r)$ parameterized by a. Property 5 can be translated into an equation to determine this parameter:

$$\lim_{r \to \rho^-} f(r) = 1 - \exp(a \cdot \rho^-) = -1 \tag{5.3}$$

After solving Equation 5.3, we get $a = \frac{\ln 2}{\rho^-}$. As f is strictly increasing, and because a is negative, f will have a maximum when $r_n = \rho^+$. To satisfy Property 5, ρ^+ must be chosen such that $0 < f(\rho^+) \leq 1$, i.e. $\rho^+ > 0$. Figure 5.8 depicts the function f used in our update rule.

As a final step, we introduce an additional hyperparameter to the update rule, namely the learning rate η. This rate aims at smoothing the evolution of the threshold θ_n, i.e. avoiding that θ_n changes too dramatically with a single query. We thus have the following update rule:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \cdot \left(1 + \eta \cdot \left(1 - 2^{\frac{r_n}{\rho^-}}\right)\right), \quad 1\right\} \tag{5.4}$$

The values of η are restricted to the range $(0, 1)$. Indeed, we still must satisfy Property 5 (a value of 1 would violate this one) and $\eta = 0$ would lead to a nonreactive system, as the threshold would never adapt. Note that this version of RAL uses the same value of η for updating both θ_n and the decision powers in α.

Choice of Hyperparameters

We acknowledge that RAL includes a non-negligible number of hyperparameters which should be well chosen in order to obtain the best results. While we do not have any rule of thumb on how to define exact values, the following guidelines help RAL learn from the streaming data:

- The initial value of θ should be high when the number of possible labels is low, to avoid that the model is always too certain about its estimation for the encountered samples

- ε should be higher when dealing with more dynamic datasets, to increase the probability to accurately grasp concept drifts; in general, we would advise using values in the range of 1 to 5%
- η should be small to avoid changing the decision powers of the different learners (α) and θ_n too abruptly; we would advise values below 0.1
- There is no specific range of values for $\rho^{\pm}$ which works better than others and these values should be picked considering the situation in which RAL is used: if unnecessary queries are a major issue, one should set ρ^{-} such that its absolute value is much higher than the one of ρ^{+}

5.3.4 Learning with a Single Classifier

RAL can also be used with a single classifier instead of a committee of learners. In that case, RAL becomes very lightweight and the only element of the system that allows it to efficiently adapt to and learn from the data stream is the variation of the certainty threshold θ by relying on the rewards r_n.

5.3.5 RAL Evaluation

To showcase the performance of RAL, we evaluate our tool and compare it to a state-of-the-art algorithm for stream-based active learning, and to random sampling (RS). In particular, we compare RAL to the Randomized Variable Uncertainty (RVU) technique presented in [158, 159] and mentioned in Section 5.1, as this approach also heavily relies on the uncertainty of the underlying ML model to take the querying decisions. We use two network-attack datasets extracted from MAWILab [135].

Data Description

Note about adoption of text from co-authors: The text in this section (Data Description) was mainly contributed by co-authors to the joint paper [C4], even though I edited it for clarity.

The traffic studied in the context of this work spans two weeks of packet traces collected in late 2015. From the labeled anomalies and attacks, we specifically focus on those which are detected consistently by all four MAWILab detectors (one detector based on PCA, one based on the Hough transform, one relying on a Gamma model, and the last one relying on the Kullback-Leibler divergence). We consider two types of attacks:

- Flooding attacks, and in particular ping flood, which consists in overwhelming a victim's machine with ICMP echo request packets (ping packets).
- Distributed network scans, and in particular UDP probing traffic. This attack consists in scanning a group of machines (i.e. sending UDP packets and analyzing the potential reply) on a usually limited number of different ports.

In the ping-flood dataset, we observe 32% of anomalous samples, while the netscan dataset includes 37% of malicious data points.

To perform the analysis in a stream-based manner, we consider a slotted, time-based approach: we split the traffic traces in consecutive time slots of ΔT seconds each, and compute a set of features describing the traffic in each of these slots (i.e. each of them corresponds to one sample). Each slot i is assigned a label $l_i \in \{0,1\}^2$, consisting of a 2-dimensional binary vector which indicates at each position j if anomaly of type $j \in \{1,2\}$ is present ($l_{i,j} = 1$) or not ($l_{i,j} = 0$) in the current time slot. The goal is to detect each of these attack types separately, with binary classifiers indicating if the attack $j \in \{1,2\}$ is present for the time slot i.

We compute a large number of features describing a time slot, using traditional packet-level measurements including traffic throughput, packet sizes, IP addresses, ports, transport protocols, flags, etc. The total set accounts for 245 features, which are computed for every time slot i. Besides using traditional features such as min/avg/max values of some of the input measurements, we also consider the empirical distribution of some of them, sampling it at many different percentiles. This provides much richer information, as the complete distribution is taken into account. We also compute the empirical entropy $H(\cdot)$ of these distributions, reflecting the feature dispersion. Appendix E describes the full set of 245 features.

Naturally, the length of a time slot ΔT influences the computation of the proposed features, and therefore the performance of the detection models. We have tried different values for ΔT and adopted $\Delta T = 5$ seconds for the computation of features, which provides a good trade-off between fine-grained temporal resolution and model performance.

Additionally, we study the variation of the statistical properties of the considered datasets, in particular detecting concept drifts in the labels. To this end, we use the Page-Hinkley test (PHT) [179], a common statistical test. Figure 5.9 depicts the cumulative number of changes observed in the ping-flood dataset, as well as the times when those changes are detected. The test detects 14 abrupt changes during the total measurement time span. The frequency of changes significantly increases in the last third of the dataset, with more than 10 changes detected in the last four days. Concept drifts occur from modifications of the underlying characteristics of the inference target and can be the reason behind sudden shifts in the performance of algorithms. The PHT results for the netscan set were highly comparable (same number of changes, and similarly distributed in time), and thus we omitted the corresponding plot.

Setup

For each benchmarked algorithm, we proceed as follows:

- First, we subdivide the considered datasets into three consecutive, disjoint parts: the initial training set, the streaming data, and the validation set. The validation set consists of the last 30% of the dataset (where there are, incidentally, most of

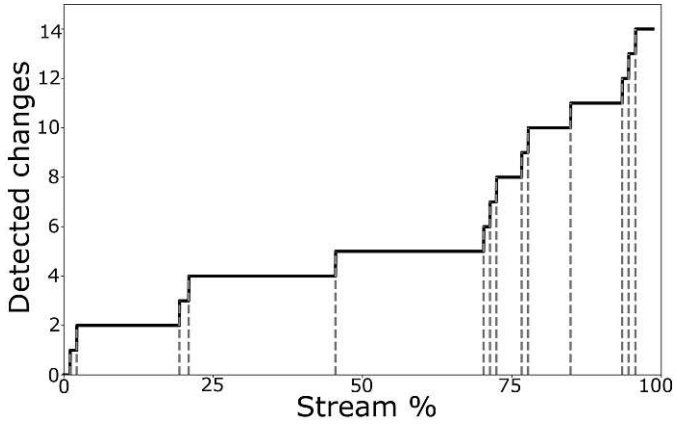


Figure 5.9: PHT detection for the ping-flood dataset; dashed lines indicate changes. © IEEE 2021

the detected concept drifts), the initial training set is a variable fraction of the first samples (varying between the first 0.5%, 1%, 2%, 5%, 10%, and 15%), and the streaming part includes all the remaining samples not belonging to the other two subsets.

- We then train a model on the initial training set and check its inference accuracy (referred to as the *initial accuracy*) on the validation part.
- Next, we run the benchmarked algorithm on the streaming part and let it pick the samples it wants to learn from. We retrain the models after each queried label.
- Finally, we evaluate the final model (i.e. trained on the initial training set and the chosen samples) again on the validation set and report this model's inference accuracy (referred to as the *final accuracy*).

In the context of this evaluation, the analysis focuses on the accuracy for the attack class (i.e., recall). We implement for both RAL and RVU the budget mechanism presented in [158], based on the ratio between the number of queries and the total number of samples observed so far; the system is allowed to issue queries to the oracle as long as this ratio is below a certain threshold, i.e. the budget. For random sampling, we use a budget indicating the exact number of samples to ask feedback for. For each dataset, we set it to the highest average number of queried samples by either RAL or RVU among all the tests with all the considered initial-training-set sizes. All tests are repeated 10 times, and we report both the average recall and its standard error. For RAL, we indicate the average number of queries performed due to the uncertainty of the underlying model and the ones issued through the ε-greedy mechanism. For RVU, we report the average number of queries. The hyperparameter values of RAL are chosen by grid search for our datasets on the training set within the ranges prescribed in Section 5.3.3. The used

ε	η	initial θ	budget	ρ^+	ρ^-
2.5%	0.01	0.9	0.05	1	-1

Table 5.6: RAL hyperparameters, selected by grid search.© IEEE 2021

values are indicated in Table 5.6. For RAL and RVU, the budget is set to 0.05 for all the experiments and we test RVU with the hyperparameters recommended in [159].

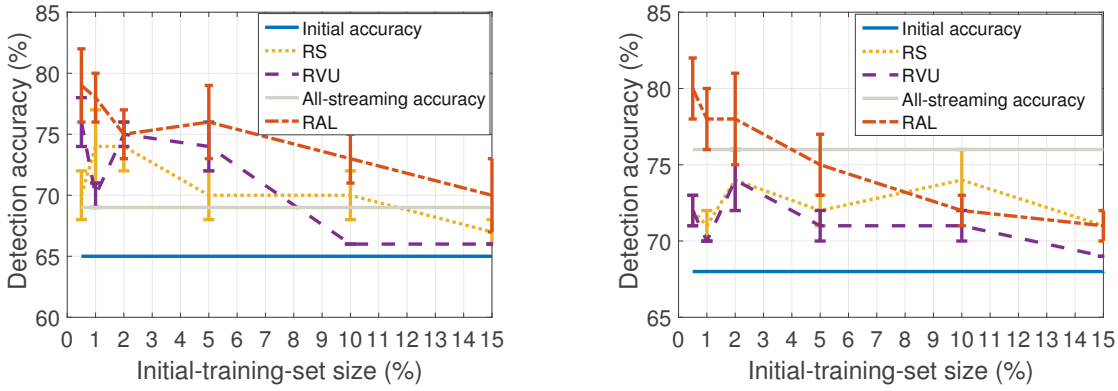
Results

The results are shown in Figures 5.10 and 5.11. The reported *all-streaming accuracy* refers to the detection accuracy obtained by the model in case it queries all the samples seen in the streaming data.

We use the committee version of RAL for our evaluations. More precisely, the committee is a voting classifier composed of three commonly used ML models that provide inference-certainty quantification, namely a k-NN model with k equal to 5, a decision tree, and a random forest with 10 trees. We also use the same model for RVU and RS. On Figures 5.10a and 5.10b, we can clearly note that RAL outperforms both RVU and RS on average. A striking example are the results for the netscan detection, where RAL obtains final accuracies which are 5 percent points higher than the ones of RVU and RS for the two smallest initial-training-set sizes. It is also worth highlighting that RAL is the only algorithm yielding higher final accuracies than the all-streaming one as long as the initial training set contains less than 5% of the data. To our surprise, RVU is often outperformed by RS. Finally, the flood-attack-detection analysis shows that the three approaches often yield a final accuracy higher than the all-streaming one, underlining that learning from the entire data stream does not necessarily output the best possible accuracy.

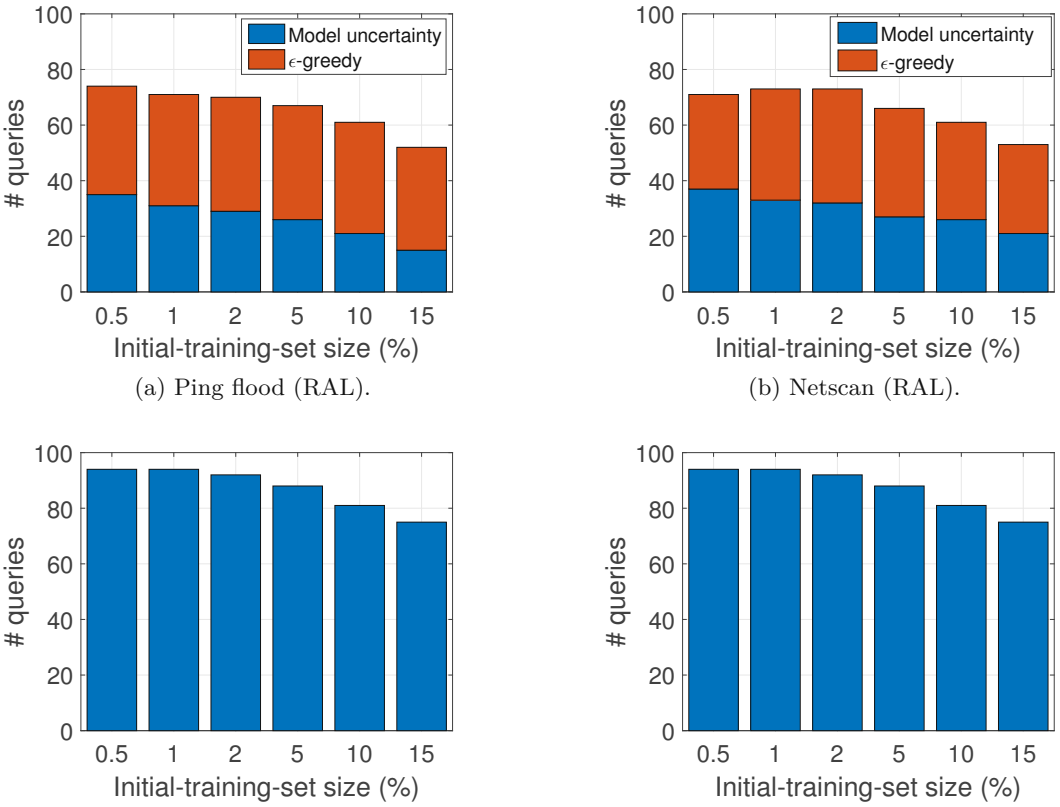
Note that the initial accuracy is constant for the two different datasets. This is due to the fact that the first 15% of these datasets consist of points with the same label (more precisely, they represent an attack).

When it comes to the number of queried samples, Figure 5.11 shows that RAL queries, on average, significantly less often than RVU – between 20% and 25% fewer queries. A non-negligible part of these queries are due to the model's uncertainty, suggesting that the samples picked by RAL for its learning purposes are wisely chosen. The results also highlight that the ε-greedy policy is very useful, as the additional exploration capability helps better deal with the concept drifts in the data, contributing to the better results shown in Figure 5.10. Finally, the number of samples/labels queried by RAL represents less than 4% of the total number of streaming samples, also showing how much one can save in terms of required labeling for training.



(a) Ping flood. (b) UDP netscan.

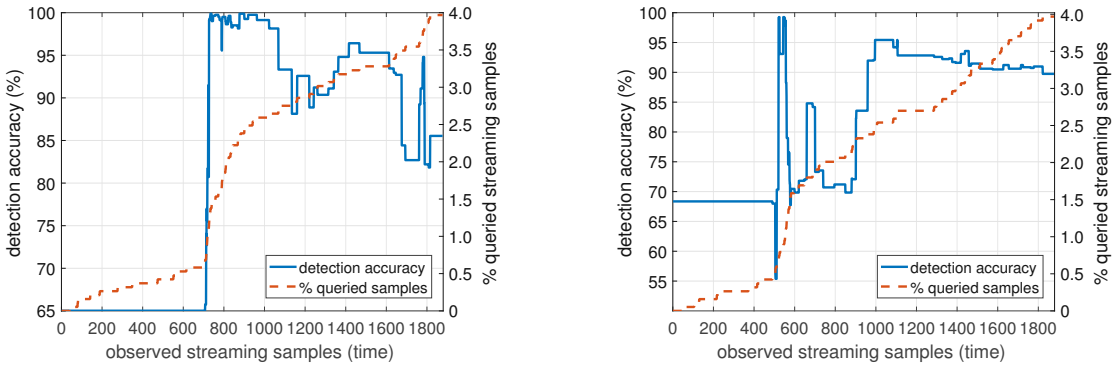
Figure 5.10: Detection accuracy for RAL, RVU, and RS. For each of the tested datasets, RAL outperforms both techniques. © IEEE 2021



(a) Ping flood (RAL). (b) Netscan (RAL).

(c) Ping flood (RVU). (d) Netscan (RVU).

Figure 5.11: Number of queries issued by RAL (top) and RVU (bottom). RAL achieves better accuracy, while querying fewer samples. © IEEE 2021



(a) Ping flood. (b) UDP netscan.

Figure 5.12: RAL's detection accuracy temporal convergence. © IEEE 2021

We also study the convergence of RAL's attack-detection performance with respect to the evolution of the streaming samples (i.e., time), for the two MAWILab attack datasets. More precisely, we evaluate RAL on the validation set after a new sample is queried. We set the initial training-set size to the first 0.5% of the data: according to Figure 5.10, such a small initial training-set provides the best results. Figure 5.12 reports the accuracy convergence for the ping-flood and netscan detection, along with the temporal evolution of the number of queried samples. We observe that the detection accuracy is not clearly converging in the two scenarios: the ping-flood-detection performance seems to converge to 90%, while there does not seem to be any convergence for the netscan case. This is not surprising, considering the fact that these datasets present multiple concepts drifts and are very dynamic. We investigated the reasons behind the sharp accuracy increase in both evaluations, and found that they are highly correlated with queries issued by the committee – not the ε-greedy scenario. This analysis confirms that acquiring the labels for which the models have a low confidence in their inference is indeed a good strategy for active learning, and in particular also for RAL. The degradation of the detection performance is likely due to the acquisition of noisy points in the dataset and to the concept drifts. The significant decrease in accuracy is mostly caused by samples queried by random exploration (ε-greedy) and not by RAL's committee, even though this mechanism also often provides performance boost by forcing RAL to explore the data space.

Based on these results, one could wonder whether the performance gain by RAL is worth the complexity of the system. Even though the accuracy gain might not be very significant, RAL's querying strategy has additional advantages over the two other techniques. For instance, RS does not take into account the uncertainty of the model nor the usefulness of the queries, meaning that there is a risk to miss interesting samples.

Indeed, querying the ground truth when the model is uncertain helps discover under-explored regions where to learn from, and RAL is additionally guided by its reward mechanism. In the specific case of the MAWILab datasets, RS would probably miss interesting attack samples, while RAL has a higher chance of querying the ground truth for these data samples and better learn how to detect attacks. When it comes to RVU, another advantage of RAL over that algorithm, besides its better performance shown above, is that the querying decisions are also influenced by the informativeness of all past queries, not only by their sheer execution; RVU does not take that information into account at all, and thus it risks querying unnecessary samples too often. This is especially problematic if querying is very expensive, or if the oracle has only limited budget/availability.

5.4 Conclusions

In this chapter, we introduced two frameworks for (adaptive) detection of network attacks: BIGMOMAL and RAL.

With BIGMOMAL – Big Data Analytics for Mobile Malware Detection –, we identified Android applications and malware on smartphones using tree-based ML models based on resource-level features with the SherLock dataset. For both of these classification targets, we obtained highly promising results in case the training sets are generated randomly. Nevertheless, those same tasks remain challenging when performed on a temporal basis as well as when relying on data instances from multiple users. There are two main takeaways from this study: firstly, our results suggest that malware-detection models using the set of features provided by the SherLock dataset can operate with very high performance only in those cases where models are tailored on a per-user basis, taking into account the statistical behavior of each user. Secondly, it seems hard to build a model that generalizes well and can cover multiple users and longer time spans, additionally due to the occurrence of concept drift.

With RAL – a novel Reinforced stream-based Active-Learning approach –, we tackle challenges of stream-based active learning, i.e. selecting the most valuable sequentially incoming samples, using reinforcement-learning principles. It does not only learn from the data stream, but also from the relevance of its querying decisions. RAL provides a completely different exploration-exploitation trade-off than existing algorithms, as it queries fewer samples of higher relevance. We showed on two MAWILab datasets that RAL provides promising results, outperforming the state of the art.

CHAPTER 6

Conclusions and Perspectives

In this thesis, we explored the field of machine learning, applied to Quality of Experience (QoE) and cybersecurity. We investigated three research questions.

Can we accurately infer video QoE from network-layer traffic despite end-to-end encryption?

We started with the inference of video QoE. In particular, we introduced two AI-based monitoring solutions, one built on top of data collected from YoMoApp, an Android application for session-based YouTube QoE measurements, and one integrated into ViCrypt, a system for real-time sensing of QoE-relevant metrics of video streaming.

In the context of the YoMoApp project, we first analyzed the video sessions of the YoMoApp users from 2014 to 2018. The data demonstrates that YouTube mobile has experienced systematic performance improvements since the beginning of the measurements, which also lead to an increase of the QoE and of user engagement. As a next step, we applied ML to the data with the goal of inferring session-based QoE metrics such as initial delay, quality-switch events, and stallings by relying exclusively on lightweight network-layer features. Our results were very encouraging for each of the considered estimation targets, especially with random forests. Indeed, the initial-delay discrimination as well as the quality-switching detection yielded a false positive rate below 5% for more than 90% of the sessions. When it comes to the inference of the re-buffering ratio, the random forest achieved an almost perfect performance for detecting bad-quality sessions with a high stalling ratio. Nevertheless, estimating the exact number of stalling events remains a very challenging task. The outcomes of our analyses pave the way for broader, non-intrusive, and privacy-preserving approach for large-scale, QoE-based monitoring of YouTube mobile.

With ViCrypt, we moved from session-based QoE-metric inference to real-time inference. More precisely, ViCrypt estimates each considered KQI every second while the user is

watching a YouTube video. To the best of our knowledge, this is the finest granularity so far used for quality inference in the context of encrypted traffic. We investigated a diverse set of ML algorithms and found, as for YoMoApp, that tree-based models were both fast, which is very important in a real-time setting, and accurate. For instance, ViCrypt output a recall close to or above 80% for each considered video-resolution class relying on the random-forest algorithm, and yielded absolute errors below 100 kbps for almost 80% of the time slots in the context of the average-bitrate inference when coupled with extremely randomized trees. Feature selection and importance analysis highlighted that a small set of lightweight network-layer features, namely the ones summarizing the characteristics of the session since the beginning of the streaming, is sufficient for achieving highly encouraging results in the context of real-time KQI inference for YouTube.

Can we accurately estimate Web QoE from network measurements? Are models trained on data coming from a specific device type applicable in a heterogeneous ecosystem?

We then moved from video to Web-QoE inference. In particular, we focused on the estimation of the RUMSI of Web sessions as well as of the QoE range with in-network traffic measurements on different device types, namely on a desktop machine, a tablet, and a smartphone. Our analysis revealed that the ML models did not generalize well across the different devices, i.e. that a model trained on data coming from a single device yielded disappointing results when applied to data coming from any different device. For example, when training and testing on data coming exclusively from a desktop device, we obtained an MAE of approximately 600 ms, which increased to more than 1000 ms when the model was trained on smartphone data instead of desktop data points. Nevertheless, we showed that the lack of cross-device generalization can be solved by properly training on data coming from a multitude of devices. Indeed, the model trained on multi-device data provided an MAE of about 650 ms for desktop samples. To the best of our knowledge, we are the first to unveil the strong impact that the device type has on the quality of Web-QoE inference. As a last step, we used ML models based on novel flow-level features, which also yielded highly promising results. In particular, a multi-device model using these kinds of features output an MAE of less than 630 ms for desktop data, which is even slightly better than the performance of the packet-level model. The conception of models based on flow-level features can be considered as another contribution of our work.

Can we rely on machine learning for security-related tasks on smartphones? In particular, how can we handle scarce resources to uncover malicious application behavior and network attacks?

After having extensively explored several topics of QoE, we shifted our focus to cybersecurity. We first introduced BIGMOMAL – Big Data Analytics for Mobile Malware Detection –, whose aim is to identify running Android applications and to detect malware activity. For this part of our work, we relied on the SherLock dataset, which contains valuable information about smartphone usage. For both inference tasks, BIGMOMAL achieved highly accurate results on a per-user basis and when the training

set was created randomly among all the data points. The accuracy for the application identification and the recall for malware detection were close to 99% for the considered users. Nevertheless, in the two cases of a training set that is chosen on a temporal basis and of a multi-user environment, accurately identifying apps and malware remains very challenging.

The last research direction we pursued was about stream-based active learning applied to the detection of network attacks. Stream-based active learning is particularly challenging, as the ML models have to rapidly decide, for each sequentially incoming sample, whether it is worth labeling or not. We proposed RAL – a novel Reinforced stream-based Active-Learning approach –, which couples active learning with reinforcement-learning principles. More precisely, the models do not only learn from the data, but also from their decisions. We evaluated RAL on two MAWILab datasets, and demonstrated that it outperformed the state of the art, not only in terms of estimation accuracy, but also in terms of the number of necessary queries, as RAL asked for significantly fewer labels than previously proposed approaches (on average 25% fewer).

Final remarks

Overall, we investigated multiple very important problems which we are confronted with in today's Internet and proposed accurate, deployable solutions for them. Our research efforts in the broad domain of AI for Networks (AI4NETS) potentially pave the way for other crucial research directions and solutions to make the Internet even more enjoyable and safer than today.

Perspectives

We tackled multiple interesting research questions during this journey. Nevertheless, there are still numerous topics worth to be explored. Some important directions that are sparked by this work are:

Move from video-QoE inference to video-QoE *prediction*: we focused on performing video-QoE estimation, i.e. we inferred the metrics after the end of a video session within the context of YoMoApp and during the current time slot with ViCrypt. A logical extension would be to predict KQIs in advance, i.e. what the video quality would look like in a couple of seconds, for instance. This would be an important step towards proactive QoE-aware traffic management. Indeed, ISPs could detect problems in advance and adjust network conditions in time to avoid QoE degradations. We initiated the discussion about this kind of management in [J2].

Generalize video-QoE inference to multiple video services: we concentrated exclusively on YouTube for both our session-based and real-time inference tasks. However, other services such as Netflix and Amazon Prime are also crucial in the world of video streaming. We could bring our studies several steps further

by generalizing YoMoApp and ViCrypt to those video providers. Building such frameworks is very challenging, as all the video providers behave differently on the network level [180], which makes it very hard to design ML approaches yielding highly accurate KQI inference for all of them. It would also be beneficial for the research community and ISPs to look deeper into QoE estimation for live-streaming services (Twitch, Vimeo, Periscope, Facebook, for instance), which become more and more popular.

Predicting root causes of video-QoE degradations: besides predicting video-QoE metrics themselves, it would be highly interesting to be able to also predict the root cause of the quality degradation (issues with the caching policies on the provider's side, network-path changes, problems with the client's video player, etc.). Indeed, identifying the degradation and its cause in advance would allow the corresponding stakeholder to remediate the situation as quickly as possible. This kind of root-cause analysis ideally makes it possible to detect problems in the video provider's network, which is not necessary visible to the ISP. First steps into this direction have been discussed in [181, 182, 183]. However, the proposed approaches do not allow to make predictions in advance, before the QoE degradation happens. While the systems presented in [182, 181] are quite lightweight, the black-box technique introduced in [183] is very complex. Moreover, none of these frameworks has seen its generalizability assessed on several different services. Our aim would be to develop automatic, lightweight, generalizable, and explainable ML systems which are able to accurately predict root causes in the context of encrypted traffic.

Advanced clustering for better Web-QoE inference: so far, we have built Web-QoE estimation models on a device-type basis, and demonstrated that they provide very promising results. Another direction we could investigate would be to rely on models tailored to the type of content provided by the Websites (images, videos, JavaScript, HTML, etc.). The rationale behind this idea is that pages hosting similar content have a high chance of behaving similarly on the network level. We could also go one step further and use clustering techniques to automatically subdivide Webpages into clusters and train ML models on top of them.

Generalize BIGMOMAL to multiple types of devices: we applied BIGMOMAL exclusively on data coming from one particular Android smartphone. However, IoT devices are on the rise, and extending BIGMOMAL to work on different devices would be a challenging yet strategically important task.

Density-based measures for RAL: RAL is so far relying on uncertainty measures to determine whether it should ask for a label or not. An interesting alternative to investigate would be density-based approaches, in a similar fashion to DBSCAN.

Semi-supervised learning for RAL: the current version of RAL is based on supervised learning. We could dive into the topics of semi-supervised learning, which combines the advantages of both supervised and unsupervised learning, in order

to attempt to improve the performance of RAL. More precisely, RAL would not only learn from the labeled data, but also from the unlabeled data points. This learning paradigm is designed such that models are able to learn from a training set composed of a small amount of labeled data and a large amount of unlabeled samples. In [184, 185], the authors explored the semi-supervised learning applied to active learning in the pool-based setting. However, RAL is tackling stream-based problems, which comes with its own set of challenges.

List of Publications

Journals

[**J5**] *Improving Web QoE Monitoring for Encrypted Network Traffic through Time Series Modeling*
N. Wehner, M. Seufert, J. Schüler, **S. Wassermann**, P. Casas, T. Hossfeld
ACM SIGMETRICS Performance Evaluation Review, vol. 48, no. 4, pp. 37-40, 2021
DOI: 10.1145/3466826.3466840

[**J4**] *Adaptive and Reinforcement Learning Approaches for Online Network Monitoring and Analysis*
S. Wassermann, T. Cuvelier, P. Mulinka, P. Casas
IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 1832-1849, 2021
DOI: 10.1109/TNSM.2020.3037486

[**J3**] *ViCrypt to the Rescue: Real-Time, Machine Learning-Driven Video QoE Monitoring for Encrypted Streaming Traffic*
S. Wassermann, M. Seufert, P. Casas, G. Li, L. Kuang
IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2007-2023, 2020
DOI: 10.1109/TNSM.2020.3036497

[**J2**] *Considering User Behavior in the Quality of Experience Cycle: Towards Proactive QoE-aware Traffic Management*
M. Seufert, **S. Wassermann**, P. Casas
IEEE Communications Letters, vol. 23, no. 7, pp. 1145-1148, 2019
DOI: 10.1109/LCOMM.2019.2914038

[**J1**] *Machine Learning Models for YouTube QoE and User Engagement Prediction in Smartphones*
S. Wassermann, N. Wehner, P. Casas
ACM SIGMETRICS Performance Evaluation Review, vol. 46, no. 3, pp. 155-158, 2018
DOI: 10.1145/3308897.3308962

Conference Papers

[**C6**] *Mobile Web and App QoE Monitoring for ISPs – from Encrypted Traffic to Speed Index through Machine Learning*
P. Casas, **S. Wassermann**, N. Wehner, M. Seufert, J. Schüler, T. Hoßfeld
Wireless and Mobile Networking Conference, October 2021
DOI: 10.23919/WMNC53478.2021.9619058
Best Paper Award

[**C5**] *Are you on Mobile or Desktop? On the Impact of End-User Device on Web QoE Inference from Encrypted Traffic*
S. Wassermann, P. Casas, Z. Ben Houidi, A. Huet, M. Seufert, N. Wehner, J. Schüler, S. Cai, H. Shi, J. Xu, T. Hoßfeld, D. Rossi
International Conference on Network and Service Management, November 2020
DOI: 10.23919/CNSM50824.2020.9269095

[**C4**] *ADAM & RAL: Adaptive Memory Learning and Reinforcement Active Learning for Network Monitoring*
S. Wassermann, T. Cuvelier, P. Mulinka, P. Casas
International Conference on Network and Service Management, October 2019
DOI: 10.23919/CNSM46954.2019.9012675

[**C3**] *On the Analysis of YouTube QoE in Cellular Networks through In-Smartphone Measurements*
S. Wassermann, P. Casas, M.Seufert, F. Wamser
Wireless and Mobile Networking Conference, September 2019
DOI: 10.23919/WMNC.2019.8881828
Best Paper Award Runner Up

[**C2**] *Beauty is in the Eye of the Smartphone Holder – A Data Driven Analysis of YouTube Mobile QoE*
N. Wehner, **S. Wassermann**, P. Casas, M. Seufert, F. Wamser
Conference on Network and Service Management, November 2018

[**C1**] Improving QoE Prediction in Mobile Video through Machine Learning
P. Casas, **S. Wassermann**
International Conference on Network of the Future, November 2017
DOI: 10.1109/NOF.2017.8251212
Best Paper Award Runner Up

Workshop Papers

[**W3**] *RAL – Improving Stream-Based Active Learning by Reinforcement Learning*
S. Wassermann, T. Cuvelier, P. Casas

European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Workshop on Interactive Adaptive Learning, September 2019

[**W2**] *I See What You See: Real Time Prediction of Video Quality from Encrypted Streaming Traffic*
S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li
Workshop on QoE-based Analysis and Management of Data Communication Networks, October 2019
DOI: 10.1145/3349611.3355549

[**W1**] *BIGMOMAL – Big Data Analytics for Mobile Malware Detection*
S. Wassermann, P. Casas
Workshop on Traffic Measurements for Cybersecurity, August 2018
DOI: 10.1145/3229598.3229600

Extended Abstracts

[**A2**] *How Good is your Mobile (Web) Surfing? Speed Index Inference from Encrypted Traffic*
S.Wassermann, P. Casas, M. Seufert, N. Wehner, J. Schüler, T. Hossfeld
SIGCOMM Posters, Demos, and Student Research Competition, August 2020
DOI: 10.1145/3405837.3411382

[**A1**] *RAL – Reinforcement Active Learning for Network Traffic Monitoring and Analysis*
S. Wassermann, T. Cuvelier, P. Casas
SIGCOMM Posters, Demos, and Student Research Competition, August 2020
DOI: 10.1145/3405837.3411390

Demo Papers

[**D1**] *Let me Decrypt your Beauty: Real-time Prediction of Video Resolution and Bitrate for Encrypted Video Streaming*
S. Wassermann, M. Seufert, P. Casas, L. Gang, K. Li
Demonstrations of the Network Traffic Measurement and Analysis Conference, June 2019
DOI: 10.23919/TMA.2019.8784589

Posters

[**P4**] *Improving Stream-Based Active Learning with Reinforcement Learning*
S.Wassermann, T. Cuvelier, P. Casas

presented during the poster session at the Women in Machine Learning Workshop co-located with the conference on Neural Information Processing Systems, December 2019

[**P3**] *Decrypting Video Quality from Encrypted Streaming Traffic*
S.Wassermann, P. Casas
accepted to the poster session at the Women in Machine Learning Workshop co-located with the conference on Neural Information Processing Systems, December 2019

[**P2**] *ViCrypt: Real-time, Fine-grained Prediction of Video Quality from Encrypted Streaming Traffic*
S. Wassermann, M. Seufert, P. Casas
presented during the poster session at the Internet Measurement Conference, Early Work, Tools, and Datasets Track, October 2019

[**P1**] *BIGMOMAL – Big Data Analytics for Mobile Malware Detection*
S.Wassermann, P. Casas
presented during the poster session at the Internet Measurement Conference, November 2017

Bibliography

[1] Kaspersky Lab, "Mobile Malware Evolution 2020," March 2021. [Online]. Available: https://securelist.com/mobile-malware-evolution-2020/101029/

[2] M. H. Mazhar and Z. Shafiq, "Real-Time Video Quality of Experience Monitoring for HTTPS and QUIC," in *INFOCOM – Conference on Computer Communications*, April 2018.

[3] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-Time QoE Detection for Encrypted YouTube Traffic," in *Multimedia Systems Conference*, June 2019.

[4] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, Detection and Analysis of Malware for Smart Devices," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKenzie, "Cognitive Networks: Adaptation and Learning to Achieve End-to-End Performance Objectives," *IEEE Communications Magazine*, vol. 44, no. 12, pp. 51–57, 2006.

[8] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[9] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014.

[10] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A Survey on Machine-Learning Techniques in Cognitive Radios," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1136–1159, 2013.

[11] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[12] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[13] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2392–2431, 2017.

[14] T. 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.

[15] J. Saxe, R. Harang, C. Wild, and H. Sanders, "A Deep Learning Approach to Fast, Format-Agnostic Detection of Malicious Web Content," in *Security and Privacy Workshops*, May 2018.

[16] G. Marín, P. Casas, and G. Capdehourat, "DeepSec Meets RawPower – Deep Learning for Detection of Network Attacks Using Raw Representations," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 147–150, 2019.

[17] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network Traffic Anomaly Detection Using Recurrent Neural Networks," *arXiv preprint arXiv:1803.10769*, March 2018.

[18] J. Zhao, S. Shetty, and J. W. Pan, "Feature-Based Transfer Learning for Network Security," in *Military Communications Conference*, October 2017.

[19] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "LEMNA: Explaining Deep Learning Based Security Applications," in *Conference on Computer and Communications Security*, October 2018.

[20] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Workshop on Hot Topics in Networks*, November 2016.

[22] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *Conference of the Special Interest Group on Data Communication*, August 2017.

[23] 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 Conference on Networks and Communications*, June 2015.

[24] 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 *Conference on Innovation in Clouds, Internet and Networks and Workshops*, February 2019.

[25] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP – Standards and Design Principles," in *Annual Conference on Multimedia systems*, February 2011.

[26] Apple, "HTTP Live Streaming," May 2009. [Online]. Available: https://developer.apple.com/streaming/

[27] ITU, "ITU-T Recommendation P.800: Methods for Subjective Determination of Transmission Quality," 1996.

[28] ——, "ITU-T Recommendation P.1203: Parametric Bitstream-based Quality Assessmentof Progressive Download and Adaptive Audiovisual Streaming Servicesover Reliable Transport." 2016.

[29] P. Meenan, "Real User Monitoring SpeedIndex (RUMSI) - SpeedIndex Measurements from the Field using Resource Timings," 2020. [Online]. Available: https://github.com/WPO-Foundation/RUM-SpeedIndex

[30] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 37–42, 2016.

[31] G. Kessler and S. Shepard, "A Primer On Internet and TCP/IP Tools and Utilities," Internet Requests for Comments, RFC Editor, RFC 2151, June 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2151.txt

[32] J. Postel, "Internet Control Message Protocol," Internet Requests for Comments, RFC Editor, RFC 792, September 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc792.txt

[33] G. Malkin, "Traceroute Using an IP Option," Internet Requests for Comments, RFC Editor, RFC 1393, January 1993. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1393.txt

[34] P. Goyal and A. Goyal, "Comparative Study of Two Most Popular Packet Sniffing Tools – Tcpdump and Wireshark," in *International Conference on Computational Intelligence and Communication Networks*, September 2017.

[35] M. Mellia, A. Carpani, and R. L. Cigno, "TStat: TCP Statistic and Analysis Tool," in *International Workshop on Quality of Service in Multiservice IP Networks*, February 2003.

[36] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," Internet Requests for Comments, RFC Editor, RFC 1157, May 1990. [Online]. Available: https://rfc-editor.org/rfc/rfc1157.txt

[37] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[38] P. Geurts, A. Irrthum, and L. Wehenkel, "Supervised Learning with Decision Tree-Based Methods in Computational and Systems Biology," *Molecular BioSystems*, vol. 5, no. 12, pp. 1593–1605, 2009.

[39] E. Fix and J. Hodges, "Discriminatory Analysis: Nonparametric Discrimination, Consistency Properties," USAF School of Aviation Medicine, Tech. Rep., 1951.

[40] H. Zhang, "The Optimality of Naive Bayes," in *International Florida Artificial Intelligence Research Society Conference*, May 2004.

[41] B. Boser, I. Guyon, and V. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Workshop on Computational Learning Theory*, July 1992.

[42] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[43] S. W. Kwok and C. Carter, "Multiple Decision Trees," *Uncertainty in Artificial Intelligence*, vol. 9, pp. 327–335, 1990.

[44] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[45] T. Ho, "Random Decision Forests," in *International Conference on Document Analysis and Recognition*, August 1995.

[46] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely Randomized Trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[47] R. Schapire, "The Boosting Approach to Machine Learning: An Overview," *Nonlinear Estimation and Classification*, pp. 149–171, 2003.

[48] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," in *European Conference on Computational Learning Theory*, March 1995.

[49] J. Friedman, "Greedy Function Approximation: a Gradient Boosting Machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[50] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *International Conference on Knowledge Discovery and Data Mining*, August 2016.

[51] D. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[52] I. Guyon, J. Weston, S. D. Barnhill, and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines," *Machine Learning*, vol. 46, no. 1, pp. 389–422, 2002.

[53] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[54] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *International Conference on Knowledge Discovery and Data Mining*, August 1996.

[55] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation Forest," in *International Conference on Data Mining*, December 2008.

[56] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 93–104, 2000.

[57] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[58] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[59] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[60] P. Auer, "Using Confidence Bounds for Exploitation-Exploration Trade-offs," *Journal of Machine Learning Research*, vol. 3, no. 11, pp. 397–422, 2002.

[61] N. Abe and P. M. Long, "Associative Reinforcement Learning Using Linear Probabilistic Concepts," in *International Conference on Machine Learning*, June 1999.

[62] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," in *International Conference on World Wide Web*, April 2010.

[63] Sandvine, "Global Internet Phenomena Report 2019," September 2019. [Online]. Available: https://www.sandvine.com/global-internet-phenomena-report-2019

[64] ——, "2020 COVID Internet Phenomena Spotlight Report," May 2020. [Online]. Available: https://www.sandvine.com/covid-internet-spotlight-report

[65] Futuresource Consulting, "The Sustainable Future of Video Entertainment," August 2020. [Online]. Available: https://www.interdigital.com/download/5fa0694a8934bfdf5f00596a

[66] R. Schatz, T. Hoßfeld, and P. Casas, "Passive YouTube QoE Monitoring for ISPs," in *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, July 2012.

[67] P. Casas, M. Seufert, and R. Schatz, "YOUQMON: A System for On-Line Monitoring of YouTube QoE in Operational 3G Networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[68] Ericsson, "Ericsson Mobility Report November 2020," November 2020. [Online]. Available: https://www.ericsson.com/4adc87/assets/local/reports-papers/mobility-report/documents/2020/november-2020-ericsson-mobility-report.pdf

[69] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2014.

[70] D. Ghadiyaram, J. Pan, and A. C. Bovik, "A Time-Varying Subjective Quality Model for Mobile Streaming Videos with Stalling Events," in *Applications of Digital Image Processing*, September 2015.

[71] K. Zeng, H. Yeganeh, and Z. Wang, "Quality-of-Experience of Streaming Video: Interactions Between Presentation Quality and Playback Stalling," in *International Conference on Image Processing*, September 2016.

[72] S. Egger, T. Hoßfeld, R. Schatz, and M. Fiedler, "Waiting Times in Quality of Experience for Web Based Services," in *International Workshop on Quality of Multimedia Experience*, July 2012.

[73] C. Timmerer, M. Maiero, and B. Rainer, "Which Adaptation Logic? An Objective and Subjective Performance Evaluation of HTTP-Based Adaptive Media Streaming Systems," *arXiv preprint arXiv:1606.00341*, June 2016.

[74] H. Ott, K. Miller, and A. Wolisz, "Simulation Framework for HTTP-Based Adaptive Streaming Applications," in *Workshop on NS-3*, June 2017.

[75] M. Seufert, N. Wehner, and P. Casas, "A Fair Share for All: TCP-Inspired Adaptation Logic for QoE Fairness Among Heterogeneous HTTP Adaptive Video Streaming Clients," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 475–488, 2019.

[76] International Standards Organization/International Electrotechnical Commission (ISO/IEC), "23009-1:2012 Information Technology – Dynamic Adaptive Streaming over HTTP (DASH) – Part 1: Media Presentation Description and Segment Formats," 2012.

[77] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner, "Assessing Effect Sizes of Influence Factors Towards a QoE Model for HTTP Adaptive Streaming," in *International Workshop on Quality of Multimedia Experience*, September 2014.

[78] M. Seufert, T. Hoßfeld, and C. Sieber, "Impact of Intermediate Layer on Quality of Experience of HTTP Adaptive Streaming," in *International Conference on Network and Service Management*, November 2015.

[79] H. T. T. Tran, T. Vu, N. P. Ngoc, and T. C. Thang, "A Novel Quality Model for HTTP Adaptive Streaming," in *International Conference on Communications and Electronics*, July 2016.

[80] F. Wang, Z. Fei, J. Wang, Y. Liu, and Z. Wu, "HAS QoE Prediction Based on Dynamic Video Features with Data Mining in LTE Network," *Science China Information Sciences*, vol. 60, no. 4, pp. 1–14, 2017.

[81] B. Staehle, M. Hirth, R. Pries, F. Wamser, and D. Staehle, "YoMo: A YouTube Application Comfort Monitoring Tool," in *New Dimensions in the Assessment and Support of Quality of Experience for Multimedia Applications*, June 2010.

[82] P. Casas, R. Schatz, and T. Hoßfeld, "Monitoring YouTube QoE: Is Your Mobile Network Delivering the Right Experience to Your Customers?" in *Wireless Communications and Networking Conference*, April 2013.

[83] H. Nam, K.-H. Kim, D. Calin, and H. Schulzrinne, "YouSlow: A Performance Analysis Tool for Adaptive Bitrate Video Streaming," *SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 111–112, 2014.

[84] H. Nam, K.-H. Kim, and H. Schulzrinne, "QoE Matters More Than QoS: Why People Stop Watching Cat Videos," in *INFOCOM – Conference on Computer Communications*, April 2016.

[85] H. Nam, H. Schulzrinne, and K.-H. Kim, "YouSlow: What Influences User Abandonment Behavior for Internet Video?" Columbia University, Tech. Rep., December 2016.

[86] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the Edge Network," in *Internet Measurement Conference*, November 2010.

[87] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao, "Mobilyzer: An Open Platform for Controllable Mobile Network Measurements," in *Annual International Conference on Mobile Systems, Applications, and Services*, May 2015.

[88] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau, "QoE Doctor: Diagnosing Mobile App QoE with Automated UI Control and Cross-Layer Analysis," in *Internet Measurement Conference*, November 2014.

[89] I. Ketykó, K. De Moor, T. De Pessemier, A. J. Verdejo, K. Vanhecke, W. Joseph, L. Martens, and L. De Marez, "QoE Measurement of Mobile YouTube Video Streaming," in *Workshop on Mobile Video Delivery*, October 2010.

[90] G. Gómez, L. Hortigüela, Q. Pérez, J. Lorca, R. García, and M. C. Aguayo-Torres, "YouTube QoE Evaluation Tool for Android Wireless Terminals," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, pp. 1–14, 2014.

[91] 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. 22 267–22 301, 2017.

[92] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, "BUFFEST: Predicting Buffer Conditions and Real-Time Requirements of HTTP(S) Adaptive Streaming Clients," in *Multimedia Systems Conference*, June 2017.

[93] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring Video QoE from Encrypted Traffic," in *Internet Measurement Conference*, November 2016.

[94] 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 Conference*, June 2018.

[95] T. Mangla, E. Halepovic, E. Zegura, and M. Ammar, "Drop the Packets: Using Coarse-Grained Data to Detect Video Performance Issues," in *International Conference on Emerging Networking Experiments and Technologies*, November 2020.

[96] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward Quality-of-Experience Estimation for Mobile Apps from Passive Network Measurements," in *Workshop on Mobile Computing Systems and Applications*, February 2014.

[97] P. Casas, R. Schatz, F. Wamser, M. Seufert, and R. Irmer, "Exploring QoE in Cellular Networks: How Much Bandwidth do You Need for Popular Smartphone Apps?" in *Workshop on All Things Cellular: Operations, Applications and Challenges*, August 2015.

[98] T. Hoßfeld, R. Schatz, E. Biersack, and L. Plissonneau, *Internet Video Delivery in YouTube: From Traffic Measurements to Quality of Experience*, vol. 7754, pp. 264–301, 2013.

[99] T. Hoßfeld, L. Skorin-Kapov, P. E. Heegaard, M. Varela, and K.-T. Chen, "On Additive and Multiplicative QoS-QoE Models for Multiple QoS Parameters," in *Workshop on Perceptual Quality of Systems*, August 2016.

[100] M. Dasari, S. Vargas, A. Bhattacharya, A. Balasubramanian, S. Das, and M. Ferdman, "Impact of Device Performance on Mobile Internet QoE," in *Internet Measurement Conference*, October 2018.

[101] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs," in *Internet Measurement Conference*, November 2012.

[102] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "YouTube QoE Estimation Based on the Analysis of Encrypted Network Traffic Using Machine Learning," in *Globecom Workshops*, December 2016.

[103] 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 *Conference on Innovation in Clouds, Internet and Networks and Workshops*, February 2019.

[104] ——, "Features that Matter: Feature Selection for On-line Stalling Prediction in Encrypted Video Streaming," in *INFOCOM – Conference on Computer Communications Workshops*, April 2019.

[105] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "I See What You See: Real Time Prediction of Video Quality from Encrypted Streaming Traffic," in *Internet-QoE Workshop on QoE-based Analysis and Management of Data Communication Networks*, October 2019.

[106] 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 *Network Traffic Measurement and Analysis Conference*, June 2017.

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

[108] T. Karagkioules, D. Tsilimantos, S. Valentin, F. Wamser, B. Zeidler, M. Seufert, F. Loh, and P. Tran-Gia, "A Public Dataset for YouTube's Mobile Streaming Client," in *Network Traffic Measurement and Analysis Conference*, June 2018.

[109] A. Strehl and M. Littman, "Online Linear Regression and its Application to Model-Based Reinforcement Learning," *Conference on Neural Information Processing Systems*, December 2007.

[110] P. P. Pébay, "Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments," Sandia National Laboratories, Tech. Rep., September 2008.

[111] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[112] Cisco, "Cisco Annual Internet Report (2018-2023) White Paper, Updated March 2020," Cisco, Tech. Rep., March 2020.

[113] D. N. da Hora, A. S. Asrese, V. Christophides, R. Teixeira, and D. Rossi, "Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-Fold Metrics," in *International Conference on Passive and Active Network Measurement*, March 2018.

[114] E. Ibarrola, I. Taboada, and R. Ortega, "Web QoE Evaluation in Multi-Agent Networks: Validation of ITU-T G.1030," in *International Conference on Autonomic and Autonomous Systems*, April 2009.

[115] ITU, "G.1030 : Estimating End-to-end Performance in IP Networks for Data Applications," 2014.

[116] T. Hoßfeld, F. Metzger, and D. Rossi, "Speed Index: Relating the Industrial Standard for User Perceived Web Performance to Web QoE," in *International Workshop on Quality of Multimedia Experience*, May 2018.

[117] Q. Gao, P. Dey, and P. Ahammad, "Perceived Performance of Top Retail Webpages in the Wild: Insights from Large-Scale Crowdsourcing of Above-the-Fold QoE," in *Workshop on QoE-based Analysis and Management of Data Communication Networks*, August 2017.

[118] A. Sackl, S. Egger, and R. Schatz, "The Influence of Network Quality Fluctuations on Web QoE," in *International Workshop on Quality of Multimedia Experience*, September 2014.

[119] A. Sackl, P. Casas, R. Schatz, L. Janowski, and R. Irmer, "Quantifying the Impact of Network Bandwidth Fluctuations and Outages on Web QoE," in *International Workshop on Quality of Multimedia Experience*, May 2015.

[120] M. Varela, T. Mäki, L. Skorin-Kapov, and T. Hoßfeld, "Towards an Understanding of Visual Appeal in Website Design," in *International Workshop on Quality of Multimedia Experience*, July 2013.

[121] M. Varela, L. Skorin-Kapov, T. Mäki, and T. Hoßfeld, "QoE in the Web: A Dance of Design and Performance," in *International Workshop on Quality of Multimedia Experience*, May 2015.

[122] S. Baraković and L. Skorin-Kapov, "Survey of Research on Quality of Experience Modelling for Web Browsing," *Quality and User Experience*, vol. 2, no. 1, pp. 1–31, 2017.

[123] A. Saverimoutou, B. Mathieu, and S. Vaton, "A 6-Month Analysis of Factors Impacting Web Browsing Quality for QoE Prediction," *Computer Networks*, vol. 164, 2019.

[124] A. S. Asrese, S. J. Eravuchira, V. Bajpai, P. Sarolahti, and J. Ott, "Measuring Web Latency and Rendering Performance: Method, Tools & Longitudinal Dataset," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 535–549, 2019.

[125] M. Rajiullah, A. Lutu, A. S. Khatouni, M.-R. Fida, M. Mellia, A. Brunstrom, O. Alay, S. Alfredsson, and V. Mancuso, "Web Experience in Mobile Networks: Lessons from Two Million Page Visits," in *The World Wide Web Conference*, May 2019.

[126] S. Ihm and V. S. Pai, "Towards Understanding Modern Web Traffic," in *Internet Measurement Conference*, November 2011.

[127] M. Trevisan, I. Drago, and M. Mellia, "PAIN: A Passive Web Performance Indicator for ISPs," *Computer Networks*, vol. 149, pp. 115–126, 2019.

[128] A. Balachandran, V. Aggarwal, E. Halepovic, J. Pang, S. Seshan, S. Venkataraman, and H. Yan, "Modeling Web Quality of Experience on Cellular Networks," in *Annual International Conference on Mobile Computing and Networking*, September 2014.

[129] P. Casas, M. Seufert, F. Wamser, B. Gardlo, A. Sackl, and R. Schatz, "Next to You: Monitoring Quality of Experience in Cellular Networks From the End-Devices," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 181–196, 2016.

[130] S. Wassermann, N. Wehner, and P. Casas, "Machine Learning Models for YouTube QoE and User Engagement Prediction in Smartphones," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 155–158, 2019.

[131] A. Nikravesh, Q. A. Chen, S. Haseley, X. Zhu, G. Challen, and Z. M. Mao, "QoE Inference and Improvement Without End-Host Control," in *Symposium on Edge Computing*, October 2018.

[132] A. Huet, A. Saverimoutou, Z. Ben Houidi, H. Shi, S. Cai, J. Xu, B. Mathieu, and D. Rossi, "Revealing QoE of Web Users from Encrypted Network Traffic," in *Networking Conference*, June 2020.

[133] A. Huet, Z. Ben Houidi, S. Cai, H. Shi, J. Xu, and D. Rossi, "Web Quality of Experience from Encrypted Packets," in *SIGCOMM Posters and Demos*, August 2019.

[134] P. Casas, J. Vanerio, and K. Fukuda, "GML Learning, a Generic Machine Learning Model for Network Measurements Analysis," in *International Conference on Network and Service Management*, November 2017.

[135] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *International Conference on Emerging Networking Experiments and Technologies*, December 2010.

[136] T. Patel, "Worldwide Smartphone Sales Soared 47% in Q4," February 2012. [Online]. Available: https://www.rcrwireless.com/20120217/devices/worldwide-smartphone-sales-soared-in-q4-of-2011-with-47-growth

[137] Gartner, "Gartner Says Smartphone Sales Surpassed One Billion Units in 2014," March 2015. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2015-03-03-gartner-says-smartphone-sales-surpassed-one-billion-units-in-2014

[138] Ofcom, "The Communications Market Report," December 2016. [Online]. Available: https://www.ofcom.org.uk/___data/assets/pdf_file/0026/95642/ICMR-Full.pdf

[139] Juniper, "2011 Mobile Threats Report," February 2012. [Online]. Available: https://www.slideshare.net/junipernetworks/juniper-networks-2011-mobile-threats-report

[140] Kaspersky Lab, "Mobile Malware Evolution 2016," February 2017. [Online]. Available: https://securelist.com/files/2017/02/Mobile_report_2016.pdf

[141] R. Unuchek, "Rooting Pokémons in Google Play Store," September 2016. [Online]. Available: https://securelist.com/rooting-pokemons-in-google-play-store/76081/

[142] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Symposium on Security and Privacy*, May 2012.

[143] S. Bhatt and P. R. Ragiri, "Security Trends in Internet of Things: a Survey," *SN Applied Sciences*, vol. 3, no. 1, pp. 1–14, 2021.

[144] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–41, 2017.

[145] L. Xue, Y. Zhou, T. Chen, X. Luo, and G. Gu, "Malton: Towards On-Device Non-Invasive Mobile Malware Analysis for ART," in *Security Symposium*, August 2017.

[146] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in *Conference on Computer and Communications Security*, October 2011.

[147] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," in *Conference on Computer and Communications Security*, October 2012.

[148] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative Security Risk Assessment of Android Permissions and Applications," in *Data and Applications Security and Privacy*, July 2013.

[149] Y. Zhou, Z. Wang, W. Zhou, and J. Xuxian, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Network and Distributed System Security Symposium*, February 2012.

[150] S. Arshad, M. A. Shah, A. Khan, and M. Ahmed, "Android Malware Detection & Protection: A Survey," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, pp. 463–475, 2016.

[151] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *Conference on Operating Systems Design and Implementation*, October 2010.

[152] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls," in *International Workshop on Security And Privacy Analytics*, March 2016.

[153] V. Afonso, F. Matheus, R. André, B. Glauco, and L. Paulo, "Identifying Android Malware Using Dynamically Obtained Features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[154] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic Framework for Android Malware Detection Using Deep Learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[155] Y. Mirsky, A. Shabtai, L. Rokach, B. Shapira, and Y. Elovici, "SherLock vs Moriarty: A Smartphone Dataset for Cybersecurity Research," in *Workshop on Artificial Intelligence and Security*, October 2016.

[156] D. T. Wagner, A. Rice, and A. R. Beresford, "Device Analyzer: Large-scale Mobile Data Collection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 53–56, 2014.

[157] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "LiveLab: Measuring Wireless Networks and Smartphone Users in the Field," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 15–20, 2011.

[158] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes, "Active Learning with Evolving Streaming Data," in *Machine Learning and Knowledge Discovery in Databases*, September 2011.

[159] ——, "Active Learning With Drifting Streaming Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 27–39, 2014.

[160] W. Xu, F. Zhao, and Z. Lu, "Active Learning Over Evolving Data Streams Using Paired Ensemble Framework," in *International Conference on Advanced Computational Intelligence*, February 2016.

[161] D. Ienco, A. Bifet, I. Žliobaitė, and B. Pfahringer, "Clustering Based Active Learning for Evolving Data Streams," in *International Conference on Discovery Science*, October 2013.

[162] B. Krawczyk, "Active and Adaptive Ensemble Learning for Online Activity Recognition from Data Streams," *Knowledge-Based Systems*, vol. 138, pp. 69–78, 2017.

[163] Y. Baram, R. El-Yaniv, and K. Luz, "Online Choice of Active Learning Algorithms," *Journal of Machine Learning Research*, vol. 5, pp. 255–291, 2004.

[164] W.-N. Hsu and H.-T. Lin, "Active Learning by Learning," in *Conference on Artificial Intelligence*, January 2015.

[165] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "The Nonstochastic Multi-armed Bandit Problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[166] G. Contardo, L. Denoyer, and T. Artières, "A Meta-Learning Approach to One-Step Active Learning," September 2017.

[167] S. Ravi and H. Larochelle, "Meta-Learning for Batch Mode Active Learning," in *International Conference on Learning Representations – Workshop Track*, April 2018.

[168] L. Song, "Stream-Based Online Active Learning in a Contextual Multi-Armed Bandit Framework," *arXiv preprint arXiv:1607.03182*, July 2016.

[169] L. Song and J. Xu, "A Contextual Bandit Approach for Stream-Based Active Learning," *arXiv preprint arXiv:1701.06725*, January 2017.

[170] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "One-Shot Learning With Memory-Augmented Neural Networks," *arXiv preprint arXiv:1605.06065*, May 2016.

[171] M. Fang, Y. Li, and T. Cohn, "Learning How to Active Learn: A Deep Reinforcement Learning Approach," *arXiv preprint arXiv:1708.02383*, August 2017.

[172] K. Pang, M. Dong, Y. Wu, and T. Hospedales, "Meta-Learning Transferable Active Learning Policies by Deep Reinforcement Learning," in *International Conference on Machine Learning – AutoML Workshop*, July 2018.

[173] P. Bachman, A. Sordoni, and A. Trischler, "Learning Algorithms for Active Learning," in *International Conference on Machine Learning*, August 2017.

[174] M. Woodward and C. Finn, "Active One-Shot Learning," in *Conference on Neural Information Processing Systems – Deep Reinforcement Learning Workshop*, December 2016.

[175] K. Konyushkova, R. Sznitman, and P. Fua, "Learning Active Learning From Data," *Conference on Neural Information Processing Systems*, December 2017.

[176] P. Casas, F. Soro, J. Vanerio, G. Settanni, and A. D'Alconzo, "Network Security and Anomaly Detection with Big-DAMA, a Big Data Analytics Framework," in *International Conference on Cloud Networking*, September 2017.

[177] S. Wassermann, P. Casas, T. Cuvelier, and B. Donnet, "NETPerfTrace: Predicting Internet Path Dynamics and Performance with Machine Learning," in *Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, August 2017.

[178] C. Watkins, "Learning from Delayed Rewards," Doctoral Thesis, King's College, England, 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[179] E. S. Page, "Continuous Inspection Schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[180] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, and N. Feamster, "Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience," *ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–25, 2019.

[181] P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby, "When YouTube does not Work – Analysis of QoE-Relevant Degradation in Google CDN Traffic," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 441–457, 2014.

[182] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, K. Papagiannaki, and P. Steenkiste, "Identifying the Root Cause of Video Streaming Issues on Mobile Devices," in *International Conference on Emerging Networking Experiments and Technologies*, December 2015.

[183] L. Bonniot, C. Neumann, and F. Taïani, "Towards Internet-Scale Convolutional Root-Cause Analysis with DiagNet," in *International Parallel and Distributed Processing Symposium*, May 2021.

[184] X. Zhu, J. Lafferty, and Z. Ghahramani, "Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions," in *International Conference on Machine Learning Workshop on the Continuum From Labeled to Unlabeled Data in Machine Learning and Data Mining*, August 2003.

[185] S. T. Kong, S. Jeon, J. Lee, H. Lee, and K.-H. Jung, "Relieving the Plateau: Active Semi-Supervised Learning for a Better Landscape," *arXiv preprint arXiv:2104.03525*, April 2021.

APPENDIX A

YoMoApp Features

This list indicates all the 275 network-related YoMoApp features we rely on for the session-based video-QoE-metric inferences.

1. arbytes_avg
2. arbytes_diffTraffic_avg_avg1
3. arbytes_diffTraffic_avg_avg5
4. arbytes_diffTraffic_avg_avg10
5. arbytes_diffTraffic_avg_avg30
6. arbytes_diffTraffic_avg_avg60
7. arbytes_diffTraffic_avg_std1
8. arbytes_diffTraffic_avg_std5
9. arbytes_diffTraffic_avg_std10
10. arbytes_diffTraffic_avg_std30
11. arbytes_diffTraffic_avg_std60
12. arbytes_diffTraffic_std_avg1
13. arbytes_diffTraffic_std_avg5
14. arbytes_diffTraffic_std_avg10
15. arbytes_diffTraffic_std_avg30
16. arbytes_diffTraffic_std_avg60
17. arbytes_diffTraffic_std_std1
18. arbytes_diffTraffic_std_std5
19. arbytes_diffTraffic_std_std10
20. arbytes_diffTraffic_std_std30
21. arbytes_diffTraffic_std_std60
22. arbytes_max
23. arbytes_min
24. arbytes_std
25. arbytes_traffic_avg1
26. arbytes_traffic_avg5
27. arbytes_traffic_avg10
28. arbytes_traffic_avg30
29. arbytes_traffic_avg60
30. arbytes_traffic_max1
31. arbytes_traffic_max5
32. arbytes_traffic_max10
33. arbytes_traffic_max30
34. arbytes_traffic_max60
35. arbytes_traffic_min1
36. arbytes_traffic_min5

37. arbytes_traffic_min10
38. arbytes_traffic_min30
39. arbytes_traffic_min60
40. arbytes_traffic_std1
41. arbytes_traffic_std5
42. arbytes_traffic_std10
43. arbytes_traffic_std30
44. arbytes_traffic_std60
45. atbytes_avg
46. atbytes_diffTraffic_avg_avg1
47. atbytes_diffTraffic_avg_avg5
48. atbytes_diffTraffic_avg_avg10
49. atbytes_diffTraffic_avg_avg30
50. atbytes_diffTraffic_avg_avg60
51. atbytes_diffTraffic_avg_std1
52. atbytes_diffTraffic_avg_std5
53. atbytes_diffTraffic_avg_std10
54. atbytes_diffTraffic_avg_std30
55. atbytes_diffTraffic_avg_std60
56. atbytes_diffTraffic_std_avg1
57. atbytes_diffTraffic_std_avg5
58. atbytes_diffTraffic_std_avg10
59. atbytes_diffTraffic_std_avg30
60. atbytes_diffTraffic_std_avg60
61. atbytes_diffTraffic_std_std1
62. atbytes_diffTraffic_std_std5
63. atbytes_diffTraffic_std_std10
64. atbytes_diffTraffic_std_std30
65. atbytes_diffTraffic_std_std60
66. atbytes_max
67. atbytes_min
68. atbytes_std
69. atbytes_traffic_avg1
70. atbytes_traffic_avg5
71. atbytes_traffic_avg10
72. atbytes_traffic_avg30
73. atbytes_traffic_avg60
74. atbytes_traffic_max1
75. atbytes_traffic_max5
76. atbytes_traffic_max10
77. atbytes_traffic_max30
78. atbytes_traffic_max60
79. atbytes_traffic_min1
80. atbytes_traffic_min5
81. atbytes_traffic_min10
82. atbytes_traffic_min30
83. atbytes_traffic_min60
84. atbytes_traffic_std1
85. atbytes_traffic_std5
86. atbytes_traffic_std10
87. atbytes_traffic_std30
88. atbytes_traffic_std60
89. mrbytes_avg
90. mrbytes_diffTraffic_avg_avg1
91. mrbytes_diffTraffic_avg_avg5
92. mrbytes_diffTraffic_avg_avg10
93. mrbytes_diffTraffic_avg_avg30
94. mrbytes_diffTraffic_avg_avg60
95. mrbytes_diffTraffic_avg_std1
96. mrbytes_diffTraffic_avg_std5
97. mrbytes_diffTraffic_avg_std10
98. mrbytes_diffTraffic_avg_std30
99. mrbytes_diffTraffic_avg_std60
100. mrbytes_diffTraffic_std_avg1
101. mrbytes_diffTraffic_std_avg5
102. mrbytes_diffTraffic_std_avg10
103. mrbytes_diffTraffic_std_avg30

104. mrbytes_diffTraffic_std_avg60
105. mrbytes_diffTraffic_std_std1
106. mrbytes_diffTraffic_std_std5
107. mrbytes_diffTraffic_std_std10
108. mrbytes_diffTraffic_std_std30
109. mrbytes_diffTraffic_std_std60
110. mrbytes_max
111. mrbytes_min
112. mrbytes_std
113. mrbytes_traffic_avg1
114. mrbytes_traffic_avg5
115. mrbytes_traffic_avg10
116. mrbytes_traffic_avg30
117. mrbytes_traffic_avg60
118. mrbytes_traffic_max1
119. mrbytes_traffic_max5
120. mrbytes_traffic_max10
121. mrbytes_traffic_max30
122. mrbytes_traffic_max60
123. mrbytes_traffic_min1
124. mrbytes_traffic_min5
125. mrbytes_traffic_min10
126. mrbytes_traffic_min30
127. mrbytes_traffic_min60
128. mrbytes_traffic_std1
129. mrbytes_traffic_std5
130. mrbytes_traffic_std10
131. mrbytes_traffic_std30
132. mrbytes_traffic_std60
133. mtbytes_avg
134. mtbytes_diffTraffic_avg_avg1
135. mtbytes_diffTraffic_avg_avg5
136. mtbytes_diffTraffic_avg_avg10
137. mtbytes_diffTraffic_avg_avg30
138. mtbytes_diffTraffic_avg_avg60
139. mtbytes_diffTraffic_avg_std1
140. mtbytes_diffTraffic_avg_std5
141. mtbytes_diffTraffic_avg_std10
142. mtbytes_diffTraffic_avg_std30
143. mtbytes_diffTraffic_avg_std60
144. mtbytes_diffTraffic_std_avg1
145. mtbytes_diffTraffic_std_avg5
146. mtbytes_diffTraffic_std_avg10
147. mtbytes_diffTraffic_std_avg30
148. mtbytes_diffTraffic_std_avg60
149. mtbytes_diffTraffic_std_std1
150. mtbytes_diffTraffic_std_std5
151. mtbytes_diffTraffic_std_std10
152. mtbytes_diffTraffic_std_std30
153. mtbytes_diffTraffic_std_std60
154. mtbytes_max
155. mtbytes_min
156. mtbytes_std
157. mtbytes_traffic_avg1
158. mtbytes_traffic_avg5
159. mtbytes_traffic_avg10
160. mtbytes_traffic_avg30
161. mtbytes_traffic_avg60
162. mtbytes_traffic_max1
163. mtbytes_traffic_max5
164. mtbytes_traffic_max10
165. mtbytes_traffic_max30
166. mtbytes_traffic_max60
167. mtbytes_traffic_min1
168. mtbytes_traffic_min5
169. mtbytes_traffic_min10
170. mtbytes_traffic_min30

171. mtbytes_traffic_min60
172. mtbytes_traffic_std1
173. mtbytes_traffic_std5
174. mtbytes_traffic_std10
175. mtbytes_traffic_std30
176. mtbytes_traffic_std60
177. nb_handovers
178. nb_network_switches
179. network_type
180. rssis_avg
181. rssis_max
182. rssis_min
183. rssis_std
184. signals_avg
185. signals_max
186. signals_min
187. signals_std
188. trbytes_avg
189. trbytes_diffTraffic_avg_avg1
190. trbytes_diffTraffic_avg_avg5
191. trbytes_diffTraffic_avg_avg10
192. trbytes_diffTraffic_avg_avg30
193. trbytes_diffTraffic_avg_avg60
194. trbytes_diffTraffic_avg_std1
195. trbytes_diffTraffic_avg_std5
196. trbytes_diffTraffic_avg_std10
197. trbytes_diffTraffic_avg_std30
198. trbytes_diffTraffic_avg_std60
199. trbytes_diffTraffic_std_avg1
200. trbytes_diffTraffic_std_avg5
201. trbytes_diffTraffic_std_avg10
202. trbytes_diffTraffic_std_avg30
203. trbytes_diffTraffic_std_avg60
204. trbytes_diffTraffic_std_std1
205. trbytes_diffTraffic_std_std5
206. trbytes_diffTraffic_std_std10
207. trbytes_diffTraffic_std_std30
208. trbytes_diffTraffic_std_std60
209. trbytes_max
210. trbytes_min
211. trbytes_std
212. trbytes_traffic_avg1
213. trbytes_traffic_avg5
214. trbytes_traffic_avg10
215. trbytes_traffic_avg30
216. trbytes_traffic_avg60
217. trbytes_traffic_max1
218. trbytes_traffic_max5
219. trbytes_traffic_max10
220. trbytes_traffic_max30
221. trbytes_traffic_max60
222. trbytes_traffic_min1
223. trbytes_traffic_min5
224. trbytes_traffic_min10
225. trbytes_traffic_min30
226. trbytes_traffic_min60
227. trbytes_traffic_std1
228. trbytes_traffic_std5
229. trbytes_traffic_std10
230. trbytes_traffic_std30
231. trbytes_traffic_std60
232. ttbytes_avg
233. ttbytes_diffTraffic_avg_avg1
234. ttbytes_diffTraffic_avg_avg5
235. ttbytes_diffTraffic_avg_avg10
236. ttbytes_diffTraffic_avg_avg30
237. ttbytes_diffTraffic_avg_avg60
238. ttbytes_diffTraffic_avg_std1

239. ttbytes_diffTraffic_avg_std5
240. ttbytes_diffTraffic_avg_std10
241. ttbytes_diffTraffic_avg_std30
242. ttbytes_diffTraffic_avg_std60
243. ttbytes_diffTraffic_std_avg1
244. ttbytes_diffTraffic_std_avg5
245. ttbytes_diffTraffic_std_avg10
246. ttbytes_diffTraffic_std_avg30
247. ttbytes_diffTraffic_std_avg60
248. ttbytes_diffTraffic_std_std1
249. ttbytes_diffTraffic_std_std5
250. ttbytes_diffTraffic_std_std10
251. ttbytes_diffTraffic_std_std30
252. ttbytes_diffTraffic_std_std60
253. ttbytes_max
254. ttbytes_min
255. ttbytes_std
256. ttbytes_traffic_avg1
257. ttbytes_traffic_avg5
258. ttbytes_traffic_avg10
259. ttbytes_traffic_avg30
260. ttbytes_traffic_avg60
261. ttbytes_traffic_max1
262. ttbytes_traffic_max5
263. ttbytes_traffic_max10
264. ttbytes_traffic_max30
265. ttbytes_traffic_max60
266. ttbytes_traffic_min1
267. ttbytes_traffic_min5
268. ttbytes_traffic_min10
269. ttbytes_traffic_min30
270. ttbytes_traffic_min60
271. ttbytes_traffic_std1
272. ttbytes_traffic_std5
273. ttbytes_traffic_std10
274. ttbytes_traffic_std30
275. ttbytes_traffic_std60

The following feature explanations are extracted from the official YoMoApp documentation[1]:

trbytes: the total number of bytes received by the device.

ttbytes: the total number of bytes transmitted by the device.

mrbytes: the number of bytes received by the device over the mobile network.

mtbytes: the number of bytes transmitted by the device over the mobile network.

arbytes: the number of bytes received solely by the application.

atbytes: the number of bytes transmitted solely by the application.

diffTraffic: the traffic difference (in terms of bytes) between two measurements.

signal: the signal strength of the mobile network.

[1] http://www.yomoapp.de/documentation.pdf

RSSI: the intensity of the received WiFi signal.

nb_network switches: the number of times the device changed its type of network connection.

nb_handovers: the number of times the device changed the base transceiver station.

APPENDIX B

ViCrypt Features

In the following list, we indicate the features used by ViCrypt for the stream-based inference of video-QoE metrics:

1. window
2. packets
3. u_packets
4. d_packets
5. packets_u_ratio
6. packets_d_ratio
7. volume
8. u_volume
9. d_volume
10. volume_u_ratio
11. volume_d_ratio
12. packets_tcp
13. packets_udp
14. packets_tcp_ratio
15. packets_udp_ratio
16. volume_tcp
17. volume_udp
18. volume_tcp_ratio
19. volume_udp_ratio
20. burst
21. u_burst
22. d_burst
23. until_first_packet
24. u_until_first_packet
25. d_until_first_packet
26. after_last_packet
27. u_after_last_packet
28. d_after_last_packet
29. throughput
30. u_throughput
31. d_throughput
32. burst_throughput
33. u_burst_throughput
34. d_burst_throughput
35. u_byte_mean
36. u_byte_var
37. u_byte_stddev
38. u_byte_cvar
39. u_byte_skew

40. u_byte_kurt
41. u_byte_min
42. u_byte_max
43. u_iat_mean
44. u_iat_var
45. u_iat_stddev
46. u_iat_cvar
47. u_iat_skew
48. u_iat_kurt
49. u_iat_min
50. u_iat_max
51. u_r_slope
52. u_r_intcpt
53. d_byte_mean
54. d_byte_var
55. d_byte_stddev
56. d_byte_cvar
57. d_byte_skew
58. d_byte_kurt
59. d_byte_min
60. d_byte_max
61. d_iat_mean
62. d_iat_var
63. d_iat_stddev
64. d_iat_cvar
65. d_iat_skew
66. d_iat_kurt
67. d_iat_min
68. d_iat_max
69. d_r_slope
70. d_r_intcpt
71. t_packets
72. tu_packets
73. td_packets
74. t_packets_u_ratio
75. t_packets_d_ratio
76. t_volume
77. tu_volume
78. td_volume
79. t_volume_u_ratio
80. t_volume_d_ratio
81. t_packets_tcp
82. t_packets_udp
83. t_packets_tcp_ratio
84. t_packets_udp_ratio
85. t_volume_tcp
86. t_volume_udp
87. t_volume_tcp_ratio
88. t_volume_udp_ratio
89. t_burst
90. tu_burst
91. td_burst
92. t_until_first_packet
93. tu_until_first_packet
94. td_until_first_packet
95. t_after_last_packet
96. tu_after_last_packet
97. td_after_last_packet
98. t_throughput
99. tu_throughput
100. td_throughput
101. t_burst_throughput
102. tu_burst_throughput
103. td_burst_throughput
104. tu_byte_mean
105. tu_byte_var
106. tu_byte_stddev
107. tu_byte_cvar
108. tu_byte_skew

109. tu_byte_kurt
110. tu_byte_min
111. tu_byte_max
112. tu_iat_mean
113. tu_iat_var
114. tu_iat_stddev
115. tu_iat_cvar
116. tu_iat_skew
117. tu_iat_kurt
118. tu_iat_min
119. tu_iat_max
120. tu_r_slope
121. tu_r_intcpt
122. td_byte_mean
123. td_byte_var
124. td_byte_stddev
125. td_byte_cvar
126. td_byte_skew
127. td_byte_kurt
128. td_byte_min
129. td_byte_max
130. td_iat_mean
131. td_iat_var
132. td_iat_stddev
133. td_iat_cvar
134. td_iat_skew
135. td_iat_kurt
136. td_iat_min
137. td_iat_max
138. td_r_slope
139. td_r_intcpt
140. c_packets
141. cu_packets
142. cd_packets
143. c_packets_u_ratio
144. c_packets_d_ratio
145. c_volume
146. cu_volume
147. cd_volume
148. c_volume_u_ratio
149. c_volume_d_ratio
150. c_packets_tcp
151. c_packets_udp
152. c_packets_tcp_ratio
153. c_packets_udp_ratio
154. c_volume_tcp
155. c_volume_udp
156. c_volume_tcp_ratio
157. c_volume_udp_ratio
158. c_burst
159. cu_burst
160. cd_burst
161. c_until_first_packet
162. cu_until_first_packet
163. cd_until_first_packet
164. c_after_last_packet
165. cu_after_last_packet
166. cd_after_last_packet
167. c_throughput
168. cu_throughput
169. cd_throughput
170. c_burst_throughput
171. cu_burst_throughput
172. cd_burst_throughput
173. cu_byte_mean
174. cu_byte_var
175. cu_byte_stddev
176. cu_byte_cvar
177. cu_byte_skew

178. cu_byte_kurt
179. cu_byte_min
180. cu_byte_max
181. cu_iat_mean
182. cu_iat_var
183. cu_iat_stddev
184. cu_iat_cvar
185. cu_iat_skew
186. cu_iat_kurt
187. cu_iat_min
188. cu_iat_max
189. cu_r_slope
190. cu_r_intcpt
191. cd_byte_mean
192. cd_byte_var
193. cd_byte_stddev
194. cd_byte_cvar
195. cd_byte_skew
196. cd_byte_kurt
197. cd_byte_min
198. cd_byte_max
199. cd_iat_mean
200. cd_iat_var
201. cd_iat_stddev
202. cd_iat_cvar
203. cd_iat_skew
204. cd_iat_kurt
205. cd_iat_min
206. cd_iat_max
207. cd_r_slope
208. cd_r_intcpt

We use the following abbreviations:

d: download.

u: upload.

c: cumulative/session.

t: trend.

tu/cu: trend/cumulative upload.

td/cd: trend/cumulative download.

APPENDIX C

Web-QoE Features

In the following list, we indicate the features used for the Web-QoE inference tasks, i.e. the features we use for the quality estimation of each Web session:

101. $CBD_{i=1...100}$, CBD features with $\Delta T = 50\,ms$
102. $CBD_{i=101...200}$, CBD features with $\Delta T = 100\,ms$
103. $CBD_{i=201...300}$, CBD features with $\Delta T = 500\,ms$

Session features:

301. web_session_duration
302. downlink_session_duration
303. uplink_session_duration
304. total_packets_down
305. total_packets_up
306. total_packets
307. total_bytes_down
308. total_bytes_up
309. total_bytes
310. mean_throughput_downlink
311. mean_throughput_uplink

APPENDIX D

BIGMOMAL Features

In the following table, we indicate the features used in the BIGMOMAL framework for the Android-application identification and malware detection. The complete explanations can be found on the Google Drive of the SherLock project[1].

Feature	Description
vsize	virtual memory size in bytes
num_threads	number of threads in this process
importance	process priority level (foreground, background, service, sleeping, etc.)
importanceReasonCode	the reason for the process' importance
importanceReasonPid	for the specified values of importanceReasonCode, the process ID of the other process that is a client of this process
start_time	the time the process started after the system boot, in clock ticks
CPU_usage	CPU utilization
lru	ordering within a particular Android priority category
cutime	time the process children have been scheduled in user mode, measured in clock ticks
utime	time the process has been scheduled in user mode, measured in clock ticks

[1] https://goo.gl/E7WiXd

Feature	Description
cstime	time the process children have been scheduled in kernel mode, measured in clock ticks
stime	time the process has been scheduled in kernel mode, measured in clock ticks
minflt	number of minor page faults
majflt	number of major page faults
cminflt	number of minor page faults caused by the children
cmajflt	number of major page faults caused by the children
nice	process priority: value in the range 19 (low priority) to -20 (high priority)
otherPrivateDirty	number of private dirty pages used by everything else besides the Dalvik and native heap
dalvikPrivateDirty	number of private dirty pages used by the Dalvik heap
nativePrivateDirty	number of private dirty pages used by the native heap
dalvikSharedDirty	number of shared dirty pages used by the Dalvik heap
nativeSharedDirty	number of shared dirty pages used by the native heap
otherSharedDirty	number of shared dirty pages used by everything else besides the Dalvik and native heap
rss	current soft limit in bytes on the Resident Set Size of the process
pid	the process ID of this process
ppid	the PID of the parent process
pgid	the ID of the foreground process group of the process
tgpid	the ID of the foreground process group of the controlling terminal of the process. -1 if the process is not connected to a terminal

Feature	Description
priority	process kernel-level priority
dalvikPss	the proportional set size for the Dalvik heap
nativePss	the proportional set size for the native heap
otherPss	the proportional set size for everything else besides the Dalvik and native heap
uidRxPackets	packets received by this application
uidTxPackets	packets transmitted by this application
uidRxBytes	bytes received by this application
uidTxBytes	bytes transmitted by this application
endcode	the address below which program code is allowed to run
startcode	the address above which program code is allowed to run
sid	the session ID of the process
guest_time	guest time of the process (time spent running a virtual CPU for a guest operating system), in clock ticks
cguest_time	guest time of the process's children, in clock ticks
exit_signal	signal to be sent to parent when we die
Itrealvalue	the time in jiffies before the next SIGALRM is sent to the process due to an interval timer. Since kernel 2.6.17, this field is no longer maintained, and is hard coded as 0
rt_priority	real-time scheduling priority, a number in the range 1 to 99 for processes scheduled under a real-time policy, or 0, for non-real-time processes
processor	CPU number last executed on

APPENDIX E

MAWILab Features

In the following list, we indicate the features describing one time slot that we used in our two MAWILab datasets.

Field	Feature	Description
Total volume	# pkts	number of packets
	# bytes	number of bytes
PKT size	pkt_h	entropy PKT
	pkt_{min,avg,max,std}	min/avg/max/std PKT
	pkt_p{1,2,5,...95,97,99}	percentiles
IP protocol	# ip_protocols	number of different IP protocols
	ipp_h	entropy IPP
	ipp_{min,avg,max,std}	min/avg/max/std IPP
	ipp_p{1,2,5,...95,97,99}	percentiles IPP
	% icmp/tcp/udp	share of IP protocols
IP TTL	ttl_h	entropy TTL
	ttl_{min,avg,max,std}	min/avg/max/std of TTL
	ttl_p{1,2,5,...95,97,99}	percentiles of TTL
IPv4/IPv6	% IPv4/IPv6	share of IPv4/IPv6 packets
	# IP_src/dst	number of unique IPs
	top_ip_src/dst	most used IPs
TCP/UDP ports	# port_src/dst	number of unique ports
	top_port_src/dst	most used ports

Field	Feature	Description
TCP/UDP ports	port_h	entropy PORT
	port_{min,avg,max,std}	min/avg/max/std PORT
	port_p{1,2,5,...95,97,99}	percentiles PORT
TCP flags (byte)	flags_h	entropy TCP flag
	flags_{min,avg,max,std}	min/avg/max/std TCP flag
	flags_p{1,2,5,...95,97,99}	percentiles TCP flag
	% SYN/ACK/PSH/...	share of each TCP flag
TCP WIN size	win_h	entropy WIN
	win_{min,avg,max,std}	min/avg/max/std WIN
	win_p{1,2,5,...95,97,99}	percentiles WIN

Index