Department of Communications and Networking

Web Quality of Experience Measurement: Metrics, Methods and Tools

Alemnew Sheferaw Asrese

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 76/2021

Web Quality of Experience Measurement: Metrics, Methods and Tools

Alemnew Sheferaw Asrese

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Electrical Engineering, Remote connection link (Zoom), on 2 July 2021 at 1400h.

Aalto University
School of Electrical Engineering
Department of Communications and Networking
Networking Technology

Supervising professor
Professor Jörg Ott, Aalto University, Finland

Thesis advisor
Dr. Pasi Sarolahti, Aalto University, Finland

Preliminary examiners
Professor Oliver Hohfeld, Brandenburg University of Technology, Germany
Dr. Pedro Casas, Austrian Institute of Technology GmbH, Austria

Opponent
Dr. Gareth Tyson, Queen Mary University of London, UK

Aalto University publication series
DOCTORAL DISSERTATIONS 76/2021

ISBN 978-952-64-0400-4 (printed)
ISBN 978-952-64-0401-1 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-64-0401-1

Unigrafia Oy
Helsinki 2021

Finland

NORDIC SWAN ECOLABEL
Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Alemnew Sheferaw Asrese

Name of the doctoral dissertation
Web Quality of Experience Measurement: Metrics, Methods and Tools

Publisher School of Electrical Engineering

Unit Department of Communications and Networking

Series Aalto University publication series DOCTORAL DISSERTATIONS 76/2021

Field of research Networking Technology

Manuscript submitted 26 May 2021 **Date of the defence** 2 July 2021

Permission for public defence granted (date) 14 May 2021 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

The web is one of the dominant applications on the Internet. Over the last three decades, the web has been evolving in terms of content types, supporting technologies, content provisioning, and access protocols. Similarly, the users' demands for fast and reliable web access have been also growing. Understanding the user browsing Quality of Experience (QoE) is of interest to content and service providers to deliver a quality service. However, the subjective nature of QoE makes it challenging to measure the web user experience on a large scale. Due to this, Quality of Service (QoS) metrics that can be measured on different layers of the web stack have been used to approximate the user experience.

In this thesis, we propose a method to calculate an objective web QoE metric that better approximates the user experience. We design and implement a measurement system and tool that can be used on a large scale. We discuss the validation of the measurement system and benchmark the system performance. We present results from measurements that have been conducted to understand the web performance and QoE both from fixed-line and cellular networks. We also discuss modeling the web QoE from the QoS metrics using existing export models (e.g., ITU-T and IQX), and machine learning algorithms (e.g., SVR, CART, BOOST).

This thesis contributes to the effort towards understanding, designing, and managing infrastructure to provide improved web QoE. Web users and content and service providers can use the methodology we have proposed and the tools we have designed to understand and troubleshoot possible bottlenecks for poor user experience. For instance, Internet Service Providers (ISPs) can deploy our tools on customer premises in their subscriber base and monitor their end-user web QoE. ISPs can use this for efficient capacity planning, network design, and web traffic management towards popular Content Delivery Networks (CDNs). The work on modeling web QoE shows that the expert models and machine learning-based models have comparable degree of performance accuracy. This thesis also shows that the expert models can accommodate new time-related metrics beyond the web latency metrics.

Keywords Web Measurement, Web Performance, Web QoE, QoE Modeling

ISBN (printed) 978-952-64-0400-4 **ISBN (pdf)** 978-952-64-0401-1

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2021

Pages 194 **urn** http://urn.fi/URN:ISBN:978-952-64-0401-1

Preface

እሴብሑ ለእግዚአብሔር ለዘረድአኒ።

The research work of this thesis was carried out at Aalto University, School of Electrical Engineering, 2014-2019. Several individuals supported me during my doctoral study. I would like to take this opportunity to convey my sincere gratitude.

First and foremost, I would like to thank my supervisor, Jörg Ott for his unreserved guidance and support provided throughout my doctoral study. I would like to express my gratitude to my advisor Pasi Sarolahti for advising me during my study. I want to thank my co-authors, Andra Lutu, Dario Rossi, Diego Neves da Hora, Ermias Andargie Walelgne, Magnus Boye, Özgü Alay, Steffie Jacob Eravuchira, Renata Teixeira, Vaibhav Bajpai, and Vassilis Christophides for their valuable contributions to this research.

I would like to thank my colleagues and support personnel at the Department of Communications and Networking for their support. My special gratitude goes to my friend and colleague Ermias for all the discussions in our study and life in general. Ermi, had not I have you as a friend at the same time as a colleague, life in Finland would have been miserable.

This research was funded partly by European Community's Seventh Framework Program – Marie-Curie Initial Training Network grant no. 607728 (METRICS), and grant no. 317647 (Leone) – and the Department of Communications and Networking. I am grateful to the project partners' team members for their suggestions and insightful discussions.

I would also like to thank my hosts during my research visits at INRIA Paris (Renta Teixeira), Technical University of Munich (Vaibhav Bajpai and Christine Lissner), and Simula Research Laboratories (Özgü Alay).

I want to thank the pre-examiners, Oliver Hohfeld and Pedro Casas for reviewing my thesis and their feedbacks. I am also grateful to Sosina

Gashaw and Yihenew Beyene for proofreading this manuscript.

I would also like to thank the EOTC clergy and congregation members in Finland, and MK Europe members. The encouragement I received from the nice and strong people at MK Europe was a motivation to stay focused and accomplish a mission.

I am immensely thankful to my families and relatives for their unconditional love and support through my school life. Last but not least, I would like to thank my better half, Ichalem, for her love and everything else.

This thesis is dedicated to my mother Tena Yihunie – who is my role model of perseverance and confidence – and Adisualem – youngest brother, who was born three weeks after this research was started.

Espoo, May 26, 2021,

Alemnew Sheferaw Asrese

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Alemnew Sheferaw Asrese, Pasi Sarolahti, Magnus Boye, Jörg Ott. WePR: A Tool for Automated Web Performance Measurement. In *Proceedings of Globecom workshops*, Washington, DC, USA, 4–8 December 2016, pages 1–6.

II Alemnew Sheferaw Asrese, Steffie Jacob Eravuchira, Vaibhav Bajpai, Pasi Sarolahti and Jörg Ott. Measuring Web Latency and Rendering Performance: Method, Tools & Longitudinal Dataset. *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, 2019, pp. 535–549.

III Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira and Dario Rossi. Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics. In *Proceedings of the 19th International Conference on Passive Active Measurement Conference*, Berlin, Germany, 26–27 March 2018, pages 31–43.

IV Alemnew Sheferaw Asrese, Ermias Andargie Walelgne, Vaibhav Bajpai, Andra Lutu, Özgü Alay and Jörg Ott. Measuring Web Quality of Experience in Cellular Networks. In *Proceedings of the 20th International Conference on Passive Active Measurement Conference*, Puerto Varas, Chile, 27–29 March 2019, pages 18–33.

Author's Contribution

Publication I: "WePR: A Tool for Automated Web Performance Measurement"

The publication is a joint work between the authors. The author was responsible for designing and developing the measurement system. The measurement dataset was collected and analyzed by the author. The author drafted and lead the paper.

Publication II: "Measuring Web Latency and Rendering Performance: Method, Tools & Longitudinal Dataset"

The publication is a joint work between the authors. The author developed the measurement system, collected the measurement and analyzed the dataset. The author drafted and lead the paper.

Publication III: "Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics"

The publication is a joint work between the authors. The author participated in developing the tool and analyzing the dataset. The author participated in writing the paper.

Publication IV: "Measuring Web Quality of Experience in Cellular Networks"

The publication is a joint work between the authors. The author designed and developed the tool. The author collected the measurement and ana-

lyzed the dataset. The author drafted and lead the paper.

List of Figures

List of Abbreviations

5G fiveth generation mobile network

ACR Absolute Category Rating

AJAX asynchronous JavaScript and XML

AMP Accelerated Mobile Page

API Application Programming Interface

AS Autonomous System

ATF Above The Fold

BGP Boarder Gatway Protocol

BOOST AdaBoost with CART

CART Classification And Regression Tree

CDN Content Delivery Network

CSS cascaded style sheets

DNS Domain Name System

DNS Domain Name System

DOM Document Object Model

FCP first contentful paint

FID First Input Delay

FID time to first input delay

FMP first meaningful paint

FP first paint

GUI graphical user interface

HAR HTTP Archive

HTML hypertext markup language

HTTP hypertext transfer protocol

IETF Internet Engineering Task Force

IP Internet Protocol

IQX exponential interdependency of the quality of experience and quality of service

ISP Internet Service Provider

ITU International Telecommunication Union

LTE Long Term Evolution

MBB Mobile Broadband

ML Machine Learning

MNO Mobile Network Operator

MOS Mean Opinion Score

PLT Page Load Time

QoE Quality of Experience

QoS Quality of Service

QUIC Quick UDP Internet Connections

RMSE Root Mean Square Error

RTT Round Trip Time

SCTP Stream Control Transmission Protocol

SIM card Subscriber Identity Module

SMUX simple multiplexing

SSL Secure Sockets Layer

SVR Support Vector Regression

TCP Transmission Control Protocol

TLS Transport Layer Security

TLS Transport Layer Security

TTCP Time to Contentfull Paint

TTFB Time To First Byte

TTFP Time to First Paint

TTI Time to Interactive

TTVC time to visually complete

URI Universal Resource Indicator

URl Universal Resource Locator

VM virtual machine

WWW Word Wide Web

XML eXtensible Markup Language

Xvfb X virtual framebuffer

1. Introduction

The world wide web [1, 2] has been an integral part of our daily lives since its birth in 1989. It remains one of the dominant applications on the Internet sharing 17% of the total Internet traffic [3]. The web was originally designed to transfer static contents such as texts and images usually hosted on a single server. The web has been evolving dramatically and has become a complex ecosystem where it contains dynamic media-rich contents such as images and scripts possibly hosted on multiple servers distributed across different administrative domains [4, 5, 6]. This evolution in the web ecosystem has not been limited to the contents, but also includes various advancements aiming at improving the web performance and the user experience and to satisfy increasing users' demand. The advancements include web building technology, transport and application protocols, and content provisioning (*i.e.,* content replication).

Nowadays, users consume the web for use in their daily lives ranging from information retrieval to information transmission, from entertainment to business communication, and from online banking to e-commerce. Moreover, the users' demand for fast, reliable, and ubiquitous web access is increasing. As a result, delivering a quality service is important to the content and service providers. This is because service quality is strongly tied to customer retention [7]. As such, the content and service providers always seek to understand end-user satisfaction with their service.

Understanding the end-user Quality of Experience (QoE) is subjective by nature, and often measured using subjective user ratings usually on an Absolute Category Rating (ACR) scale [8]. Obtaining the users' opinions about their browsing experience is limited in scale as it is resource and time consuming [9]. Internet practitioners have been using objective metrics such as Time To First Byte (TTFB) and Page Load Time (PLT). These metrics can be captured at the network and application level, respectively.

The metrics are collected either by listening the traffic flow passively and/or injecting an active traffic flow to the network. However, the metrics that pinpoint the occurrence of a particular event in the browsing session may not fit to approximate the human cognitive perception [10]. Alternative metrics such as Above The Fold (ATF) time [11] and Speed Index [12] are proposed to better proxy the user QoE. These metrics integrate all the events of the waterfall representing the visual progress of the web page. These objective metrics can be used as an input to infer the user QoE using mathematical models. Research towards finding suitable objective metrics to better approximate and understand the user browsing experience, designing the methodology, and implementing the appropriate measurement tools to capture the metrics on a large scale is essential.

1.1 Objectives and Scope

This thesis focuses on understanding web performance and QoE. The first objective of the thesis is to devise a methodology to compute web QoE metrics at the application layer without the need for user intervention. The second objective of the thesis is to propose and validate metrics to better approximate the web QoE that can be used on a large scale. One way to validate the metrics is by collecting a measurement dataset augmented with user opinions. The third objective of the thesis is to design and build a measurement system and tools that can be employed to measure web performance and QoE. The thesis studies the web performance and QoE using a longitudinal dataset collected from distributed vantage points connected to fixed-line and cellular networks. In doing so, the thesis focuses on measuring popular websites using existing measurement platforms that span large geographic areas across the globe. In this regard, the scope of the thesis is not to cover a large set of websites, but instead to focus on popular websites and measure the key performance and QoE metrics on a large scale. Moreover, web performance is affected by the protocol used in the transport and application layer. Comparing the web performance and the user browsing experience while using different protocols is considered beyond the scope of this thesis. The fourth objective of the thesis is to model the user browsing experience using state of the art mathematical and machine learning models. To model the web QoE, the thesis uses objective metrics that can be captured from different layers of the web stack using the aforementioned method and tools.

1.2 Methodology

We employed various research methodologies to achieve the objectives of this thesis. In particular, we followed design and abstraction research paradigms [13]. The design paradigm is based on the engineering principles of the system construction and iterates the steps of requirement analysis, specification, system design and implementation, and finally testing the system. This methodology was used to design the measurement system and tool in Publications I – IV. The abstraction (experimentation) method focuses on the investigation of the phenomena through a number of iterations based on hypothesis formulation, model construction and prediction, design experiment and data collection, and finally analyzing results. This methodology was applied in Publication III, where we hypothesized that the web QoE can be better approximated with the ATF time than the simple latency metrics such as the first byte arrival time. We first used state of the art web latency metrics and modeled the QoE. Then, we set up a measurement experiment and asked users to browse a set of websites and provide their opinions on their browsing experience. Using the design paradigm described above, we designed a tool to calculate the ATF from the browser while the users were surfing the web. Finally, we analyzed the collected dataset and constructed a web QoE model. As we hypothesized, we found that the ATF time better approximated the user QoE.

Reproducibility is a cornerstone of the success of scientific research. To emphasize scientific reproducibility [14], all the tools [15, 16, 17] and collected datasets along with the associated analysis script used in this thesis are made publicly available to the research community [18, 19]. Publication III received the best dataset award in the ACM Passive and Active Measurement (PAM) conference.

1.3 Summary of the Publications

This thesis is a compilation of four research publications based on original work by the author. In Publication I, we proposed a methodology and designed a measurement system to measure web performance on a large scale. In addition to measuring the latency metrics, we demonstrated a method to measure the web page rendering time. In Publication II, we analyzed the performance of three popular web pages over a 3.5-year long period from 182 vantage points (85% are connected to residential networks).

The measurement points covered 70 Autonomous Systems (ASs) across the globe. Moreover, we validated the measurement system and benchmarked the performance of the system we presented in Publication I. We analyzed a nine-month long dataset collected with the measurement system. The longitudinal dataset shows that the Domain Name System (DNS) lookup time has improved over time. It also shows that content cache deployments within the ISP's network that lower the IP path lengths towards the server help to improve the download time of websites. The nine-month long dataset of the rendering performance of four selected websites shows that the download time of the websites is twice longer than the time required to render the visible portion of the websites in half of the measurements. Moreover, we discussed the implications of our work on network and service management.

In Publication III, we designed a lightweight tool to calculate the website's ATF time using a browser resource timing Application Programming Interface (API). Furthermore, we collected and analyzed 3,400 web browsing sessions annotated with QoS metrics and user ratings on the scale 1 to 5. We contrasted domain expert QoE models (such as ITU-T and the IQX hypothesis) fed with single QoS metrics into machine learning-based models trained using the ground truth over the dataset that contains several QoS metrics as a feature. We showed that the expert models and the machine learning-based models have comparable QoE estimation accuracy. We also concluded that the expert web QoE models can accommodate new time-related metrics beyond the PLT.

In Publication IV, we combined the methodologies we presented in Publications I – III to design a command-line-based measurement tool that can be used to compute the web latency and QoE. We used the *WebLAR* tool to collect two-week long web performance measurements from 128 nodes connected to six operational Mobile Broadband (MBB) networks. Our analysis in the two-week-long web measurement shows that the DNS lookup time and Transmission Control Protocol (TCP) connection establishment time of websites vary significantly between Mobile Network Operators (MNOs). It also shows that there is no significant difference in the PLT and ATF time of websites between different MNOs. Moreover, user mobility has a small impact on the ATF time of the websites.

1.4 Contributions of the Thesis

The main contributions of this thesis are as follows:

- We have designed and implemented two methods to calculate the ATF time, which is a metric that better approximates the user web QoE. The two methods to calculate the ATF time are 1) based on comparing the pixel-wise changes in the browser's current viewport, and 2) based on a heuristic approach using the browser's resource timing API. These approaches help measure the user browsing experience at scale using an objective metric that can be collected using active measurements.
- *WePR*: a distributed measurement system that helps measure web QoS and QoE metrics on a large scale. The system has multiple components, where one of which was deployed as part of the SamKnows measurement suite[1] to capture the browsing experience from home Internet router. The other components of the system are located in a data center and calculate application-level QoE metrics by recreating web pages based on the measurement collected by the SamKnows probe. The measurements collected at the home gateway are important in understanding the web QoE on a large scale from a realistic scenario.
- *Approximate Above The Fold*: a plugin for chromium-based browsers that helps approximate the web QoE using the browser resource timing information. The plugin calculates the ATF time by integrating the download times of the images located under the current viewport of the web pages, HTML, style sheet, and script files. The author of this thesis contributed to the design and implementation of the plugin, while Diego da Hora took the lead role.
- *WebLAR*: a command-line-based web performance measurement tool that enables capturing both web QoS and QoE metrics. The tool implements both the pixel-wise comparison and browser heuristic-based ATF approximation methods. The tool was built for measuring the web from Linux measurement boxes connected to commercial cellular networks. However, the tool can be used on any platform that supports virtualization technologies.
- We provide open datasets and the associated scripts used for analysis

[1] SamKnows is a global Internet measurements platform, `www.samknows.com`

to the community. The first dataset is a longitudinal (Jan 2014 – Jul 2017) performance measurement of three popular websites from 182 probes distributed across the globe. This dataset covers mostly web latency metrics. We also provide a two-week long dataset that contains web QoS and QoE metrics from 128 measurement probes connected to cellular networks. The third dataset we provide gives a ground truth measure of web QoE from QoS metrics annotated with user ratings.

- We present models for predicting the subjective user QoE from an objective metric. We use a dataset that contains both objective QoE metrics annotated with user ratings as a ground truth. We validate the state of the art mathematical models and machine learning algorithms using our dataset by feeding individual QoS metrics as input and several QoS metrics as a feature set, respectively.

1.5 Structure of the Thesis

The thesis is structured as follows. Chapter 2 provides the necessary background on Internet measurement, the evolution of the web, web performance QoE, and methods and tools of to measure web performance QoE. In Chapter 3, we discuss the metrics, methods, and tools for web performance and QoE. The chapter describes the pixel-wise comparison and browser heuristic-based approximation methods of ATF time. Furthermore, this chapter provides the design, implementations, and validations of the measurement system and tools.

In Chapter 4, we present the web latency in fixed-line networks using a 3.5-year-long dataset and web QoE nine-month-long dataset. We start by describing the datasets and discuss the web latency for selected popular web pages. The chapter covers the longitudinal view web latency, the impact of cache presence and broadband speed in web latency, and the difference in web latency between regions and service providers, as published in Publication II. Furthermore, in this chapter, we study the web rendering performance using a nine-month-long dataset, as described in Publications I and II.

Chapter 5 studies web QoE in cellular networks. The chapter presents an analysis of a two-week-long dataset collected from 128 measurement nodes connected to one or more operational MBB networks. It covers a

comparison of the IP path length for selected popular websites between fixed-line and LTE networks. The chapter also discusses web QoE across different MNOs. Moreover, it discusses the impact of mobility on the web QoE as described in Publication IV.

Chapter 6 deals with web QoE modeling. The chapter presents a ground truth dataset covering several web QoS metrics annotated with user ratings. It discusses modeling the QoE using the state of the art expert models taking individual QoS metrics, and machine learning algorithm feeding several QoS metrics as a feature. Finally, Chapter 7 concludes the thesis and provides future research prospects.

2. State of the Art of Web Performance and Quality of Experience (QoE)

This chapter discusses the state of the art of web performance and web QoE measurement. It starts by providing a general overview of Internet measurement (Section 2.1) and then dives deeper into web application measurement. We further introduce the evolution of the web (Section 2.2) and the web performance (Section 2.3). We provide an overview of Quality of Experience (QoE) focusing on the case of web services (Section 2.4). Finally, in Section 2.5, we present the methods and tools to measure the performance and QoE of web applications.

2.1 Measuring the Internet

As the complexity and the traffic size of the Internet are growing, understanding its architecture and behavior has become more important [20]. Network measurement is the way to know and understand how the Internet is being used and how the network behaves and evolves [21]. Since the early days of the Internet, measuring the network has been a cornerstone to understanding network behavior and finding possible root causes for network problems [22]. It is also essential for network diagnostics and troubleshooting, protocol debugging, workload characterization, and performance evaluation and improvement [23]. Measurement data collected from different layers of the Internet stack help to maintain its openness and continue the evolution of the Internet [24]. The measurements are useful not only to service and content providers, but also for researchers for performance tuning, monitoring and troubleshooting, resource planning, design and development of new technologies, and for controlling malicious behavior. The measurements can also provide input for policymakers and regulators. Originally, the Internet architecture and the protocols were not designed for the purpose of measurement [20]. Gradually, different

stakeholders in the Internet ecosystem have given attention to measuring the Internet and different applications.

The key challenges of measuring the Internet are scale, opacity, and the ethics of the measurement [24]. Since the Internet is expanding rapidly and has variable behavior, it is hard to decide the size of the measurement that represents the system. Besides, designing a tool and selecting a measurement vantage point is difficult. The term vantage point refers to not only a physical location such as a node or a link, but also a network layer, network protocol, and an application [25] where a measurement is performed. For instance, a tool designed to measure web server performance is not effective for measuring the performance of other applications [20]. Moreover, as the complexity of the Internet is growing, designing the measurements and interpreting the results are challenging. The Internet is prevalent in our daily lives and the ethical considerations of the measurements and experiments are important. Thus, during the measurement process, special care needs to be taken. For instance, measurements should not breach user privacy, disrupt user communication by triggering high volume traffic on the network, or increase the load on the network and applications.

There are different Internet measurement methods: active, passive and hybrid. The following subsections describe each measurement method in detail.

2.1.1 Passive Measurement

Passive network measurement is an approach where a measurement device is attached to the network to record all or certain properties of all or a subset of the traffic that passes through the network without introducing any additional traffic. The user generated traffic is collected and analyzed without altering the packet stream. In the case of passive network measurement methods, multiple observation points might be necessary to assess the performance of the network from different vantage points. For instance, this may be necessary to assess the latency of the packet transfer between two observation points [26].

An example of passive network measurement is that Internet Service Providers (ISPs) may deploy a measurement tool and monitor the traffic that passes through the network and capture the packet traces. In passive measurement, the measurement does not have an impact on the network performance as no additional traffic is injected into the network. One of

the challenges for this kind of measurement approach is the volume of the collected data. The measurement may incur high costs for data collection, storage, and processing [27].

2.1.2 Active Measurement

In the active network measurement approach, the measurement device itself sends probing traffic into the network and measures its characteristics. In the active measurement method, the source and destination of the packets are known a priori [26]. `Ping` and `traceroute` are examples of active network measurement tools [27]. The main challenges in active network measurement are:

- The probing traffic used in the measurement may introduce additional overhead to the network.
- The performance of the network may degrade, as the additional traffic injected into the network consumes network resources. Consequently, the measurement result may also be affected.

2.1.3 Hybrid Measurement

This measurement approach combines both the active and the passive techniques to capture active metrics, passive metrics, or new metrics derived from prior knowledge [26, 22]. In the case of the hybrid approach, as in the active measurement, a packet is sent into the network and then the network characteristics and properties are monitored using a passive traffic collection method. For example, consider an anomaly detection system which relies on measurement data captured using active tests (such as `traceroute`) and traffic traces flowing in a link capture using passive listening [22]. The active probing and the passive probing work together and export the measurements to a data repository.

In this thesis, we use the active network measurement approach to understand the web performance and end-user browsing experience. In doing so, we develop active measurement tools that can be deployed in a distributed locations and crawl the web pages.

2.1.4 Internet Measurement Platforms

Internet measurement platforms are infrastructures dedicated to deploying measurement tools that apply either active or passive methods [28].

There are proprietary and public measurement platforms. The former are used inside organizations such as ISPs and content providers. Public measurement platforms are deployed and managed by different organizations and communities for various purposes and are open to the community to use them. Bajpai *et al.* [28] surveyed the available infrastructures for Internet measurement. In this section, we discuss only some of the available public measurement platforms that we could potentially leverage to perform active measurements and achieve the objectives of the thesis.

SamKnows. *SamKnows* [29] is a company specialized in installing dedicated measurement probes to monitor broadband access performance. Established in 2008, in collaboration with ISPs and regulators, SamKnows has deployed over 70,000 measurement boxes (devices) across the globe. The measurement probes are located in the access network and behind the routers of residential networks. SamKnows uses off-the-shelf hardware white-boxes with an OpenWRT firmware.

The SamKnows devices measure the network performance using various metrics including last-mile latency, forward path, end-to-end latency, upstream and downstream throughput and goodput, network availability, and application metrics including webpage download time, DNS resolution, video steaming performance, and file storage and transfer [28]. The devices usually measure towards specific servers such as MeasurementLab servers [30]. The devices select the closest measurement server by examining the Round Trip Time (RTT). SamKnows has also mobile app measurement tool to test mobile and Wi-Fi connections.

MONROE. *MONROE* is Europe-wide, open access mobile measurement platform that enables experimenters to measure the performance of Mobile Broadband (MBB) networks in a heterogeneous environment. The platform has stationary (*e.g.,* deployed in volunteers' homes) and mobile (*e.g.,* deployed on public transport) nodes distributed across Europe (mainly in Italy, Norway, Spain and Sweden). The MONROE nodes have one or more cellular connectivity Subscriber Identity Modules (SIM cards) with a commercial cellular subscription and Wi-Fi connectivity. This enables the experimenters to test the performance of different applications in a multi-homing scenario [31]. The MONROE nodes are flexible and powerful devices which enable resource-intensive applications such as video streaming to be run. The nodes have an accelerated processing unit (APU) with AMD 1GHz dual-core 64 bit processor and 4GB DRAM.

The main features of the MONROE platform are multi-homing, large

geographical coverage, mobility, flexible and powerful nodes, context-rich information about the network and location, and open access. The platform comprises 450 measurement nodes, of which 150 nodes are deployed on buses, trucks and trains. The platform gives access to metadata about the network and the location such as cell ID, signal strength, and radio access technology. The platform enables external users to access the system and deploy experiments. The platform has a web graphical user interface (GUI) and command line interface for submitting experiments, a back-end scheduler responsible for scheduling the experiments, result collection, and data storage and visualization.

MONROE nodes use docker [32] as a virtualization technology. Each experiment is required to be containerized in a docker image. This helps the experiment to run on the same devices and use the same resources. Moreover, this ensures that the user experiments run in the nodes without any interference. As such, each experiment needs to be packaged as a docker image.

SpeedChecker. *SpeedChecker* [33] is company that provides a software-based measurement platform where measurement software is deployed in the users' devices. SpeedChecker does not distribute hardware devices. Instead, it designs and develops a software-based measurement platform called 'ProbeAPI' and provides to volunteers. Volunteers install ProbeAPI on their devices so that the experiments can run on their devices in residential networks. This helps achieve a large coverage. The volunteers who host ProbeAPI are not necessarily online all the time, which results in limited availability of the ProbeAPI. The ProbeAPI allows running tests such as DIGDNS, ping, traceroute, webpage loading, HTTP GET, video steaming, and cloud performance (*e.g.,* the latency from the location of the ProbeAPI). The ProbeAPI runs on Android devices, personal computers, and embedded devices like WiFi routers. Experimenters are required to have a SpeedChecker credit to access the API and to run measurements on ProbeAPI installed on volunteers devices.

CAIDA Ark. *Ark* is a distributed measurement infrastructure primarily for active measurement [34]. Its measurement hardware called *Ark Probes*, is based on Raspberry Pis, distributed across geographically diverse locations. The Ark probes are hosted by volunteers on different network types including residential, corporate, and access networks. The default tests that run on Ark probes include TCP behavior inference (Tbit), DNS health, IPv4 and IPv6 stability, middlebox policy and transport evolution. The

infrastructure also provides an interface to run on-demand measurements.

RIPE Atlas. It is a global network of probes that actively measures Internet connectivity and reachability [35]. The probes are distributed globally and managed by RIPE NCC [36]. Atlas probes run open-source firmware. The probes are connected to volunteers' network and constantly perform built-in measurements such as ping, traceroute, SSL and DNS queries to the root name servers and also limited HTTP measurements. The measurements do not look into traffic and content, and no passive measurements are running. Measurements are scheduled by centralized command servers via reverse ssh tunneling. Fore security reason the probes do not have any open ports, they only innitiate connections [37]. RIPE NCC collects all the data produced by the probes and provides Internet maps, tools and visulizations based on the agregated data. RIPE does not allow users to perform their custom measurements on the Atlas probes. As of January 2021, more than 11,000 Atlas probes are up and actively running measuremnts.

There are measurement platforms that are a distributed set of servers hosted at research institutes such as MeasurementLab [30] and PlanetLab [38]. There are also crowdsourcing tools including NetRadar [39] and Meteor [40], or infrastructures that provide controlled and emulated measurement devices in the laboratories such as Phone-Lab [41] and PacketLab [42]. However, these platforms are not suitable for achieving the objectives of this thesis. Hence, these platforms and tools are not discussed here. That is mainly because of two reasons. First, the measurement vantage points of infrastructures such as PlanetLab do not represent most of the Internet users. Second, crowdsourced platforms such as NetRadar and Meteor do not allow external users to run a custom test.

In this thesis, we use the SamKnows and the MONROE platforms to conduct our active measurements. We use SamKnows nodes to conduct longitudinal web measurements on fixed-line networks. We leverage the MONROE platform to perform web QoE measurement on MBB networks.

2.2 Evolution of the Web

Hypertext is a way to link documents from one to an other providing a dynamic access to the linked information. Although the term hypertext predates back to 1965, in 1989 Tim Berners-Lee proposed using the concept of linking one text from the other and navigate them in software called a

browser [1]. The network of the links between the texts (also known as nodes) is called the web [2]. The nodes can be of other media types such as images, sound files, or animations. For these kinds of media files, the term hypertext expanded to hypermedia. In his proposal, Berners-Lee's main objective was to link various information in addition to maintaining a list of the documents, provide a search feature for the linked documents using keywords. To achieve this, he proposed to provide a simple protocol for exchanging the information stored in remote systems. The proposal became the Word Wide Web (WWW), simply referred to as the web.

The web has evolved from a simple static content such as text and image delivery platform to a complex ecosystem with media-rich contents. The delivery mechanism has also changed from using a single server to a number of servers per webpage usually managed by different ASs [4]. Over the last three decades, there have also been big changes in the fundamental technologies and protocols that are used in the web content delivery. Figure 2.1 shows the timeline of the major technological changes in the evolution of the web. In this section, we discuss the changes that have happen in the content, the content provisioning, and the web stacks.

2.2.1 Content Evolution

Since the inception of the web in the early 1990's, the contents of the web have changed drastically. Initially, the web was designed in 'read-only mode' to read and deliver a static content such as texts and images from a remote system to a client. In addition to supporting a 'read-write mode' – in which, the client can both read and write information to and from a remote system – now the web contains dynamic, interactive, and media-rich contents. To support the user requirements for an interactive and dynamic web, various extensions have been added to the initial specifications of the web building technologies. The prominent extensions to the web include the introduction of JavaScript in 1995, cascaded style sheets (CSS), flash and eXtensible Markup Language (XML) in 1996, web forms and asynchronous JavaScript and XML (AJAX) in 1999, `XMLHttpRequest` in 2006, and HTML 5 in 2008. The introduction of these technologies to the web has changed the original web in different ways. These extensions enable the creation of dynamic and interactive web contents. They also help to perform operations at the client side and allow the user to submit data to the server for processing.

Furthermore, the content of the web has changed over time. A study

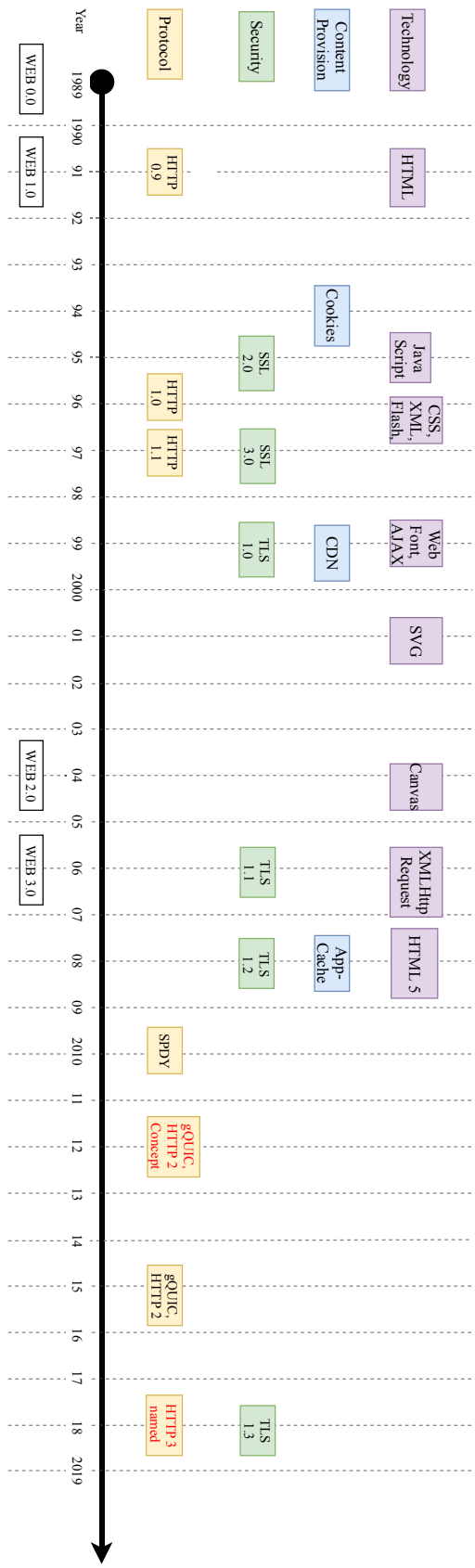

Figure 2.1. The timeline of the major technological advancements in the evolution the web (1989 to 2019). The box in red shows where the protocols are proposal time.

by Tim Bray in 1996 shows that the average sizes of the webpages were 6KB to 7KB (median 2KB) [5]. The study also shows that even though the web pages dominantly contained text type objects, more than half of them contained at least one image. Authors in [4] studied the web traffic evolution (from 1995 to 2003) and their results show that the HTTP request size has been steadily increasing. Moreover, the web page complexity as expressed by the number of the objects per page and the number of unique servers providing the contents has increased.

Fetterly *et al.* [43] crawled 720,000 webpages and studied how often the web changes, and how the changes are correlated with the characteristics of the web pages. They showed that 40% of the web pages changed within a week and 23% of the web pages that fell in the `.com` domain changed daily. The study showed that the changes are more frequent for larger webpages. Callahan *et al.* [44] analyzed the evolution of the web traffic using three-and-a-half year (2006 to mid-2009) long dataset. They showed that the average size of HTTP `GET` and `POST` transactions increased over time during the course of their dataset. This was mainly because of the introduction of web 2.0 and users increasingly downloading richer content. The authors showed that the median `GET` and `POST` transactions remained fairly constant and small over the study period. Ihm and Pai [45] performed a longitudinal analysis (from 2006 to 2010) of the browsing behavior of 70,000 web users from 187 countries. The authors showed that a heavy adoption of AJAX and Flash videos increased the size of the webpages.

Newton *et al.* [46] developed a methodology to analyze trends in the HTTP request and response sizes from web traffic. Using a dataset collected for 13 years, the authors showed that for smaller request sizes (below 1500 bytes) the changes over the years were low. Instead, in most of the cases, the HTTP responses over the years showed an increasing trend. They also showed that the connection duration by the browser had increased over time. Johnson *et al.* [6] studied the trends in desktop and mobile versions of web pages using a two-year-long dataset. The authors showed that, over the years, the number and size of the web objects (specially images and scripts) had increased.

In general, the evolution in the contents of the web pages is not limited to the primary contents of the web page but also third-party contents [47]. The size of the web pages has increased from few kilobytes (until the mid 1990's) [5] to hundreds of kilobytes (1995 to 2003) [4] to a few megabytes (2010 to 2020, as shown in Figure 2.2). Figure 2.2 shows the evolution of

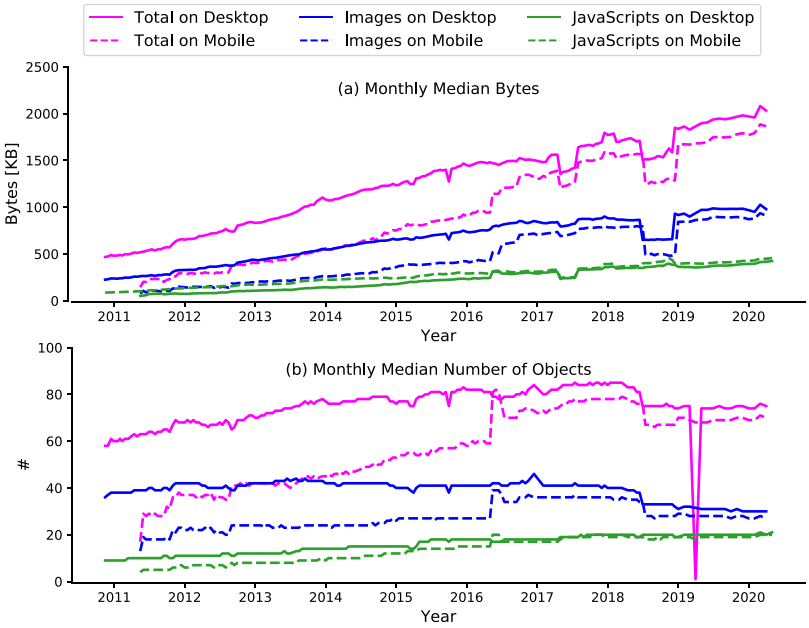

Figure 2.2. The evolution of the web content from 2010 to 2020. Data source: HTTP archive [48]

the web content in terms of the median bytes size and number of objects over the last decade (2010 to 2020) both for mobile and desktop devices. The data was collected from the HTTP archive [48]. In the data collection, the object type is identified using the MEM-type of the request. The results show that the median total number of bytes and the bytes of image types of objects have mostly been increasing. The result does not show a clear trend in the number of objects per web page over the decade. The result show that in 2017 and 2018 the monhtly median byte of the objects has dropped. We did not find a clear reason for this drop in the total and images bytes. We speculate that this may be an error in the measurement.

2.2.2 Content Provisioning

To maintain high availability and low latency, content replication is prevalent on the Internet. To this end, Content Delivery Networks (CDNs) have become a paramount component of the web [49]. The adoption of CDNs for web content delivery is drastically increasing.

Ager *et al.* [50] studied the classification of web content hosting and delivery infrastructure. They proposed an automated approach to identify the hosting infrastructure using DNS measurements and Boarder Gatway Protocol (BGP) routing table snapshots. The study showed that popular

contents are exclusively served from specific regions and ASs. Cadler *et al.* [51] analyzed millions of client-side measurements from the Bing search service to understand the performance of anycast versus unicast CDN selection method. Anycast is a technique that allows network providers to advertise the same IP prefix for multiple geographically distributed machines and the clients requests to the services are redirected to the closest machine [52]. In contrast, unicast restricts an IP address to be associated with a single machine (service end-point) [53]. The authors showed that usually the anycast mechanism performs better despite its lack of centralized control because anycast is not latency aware nor has information about the load on the server. In the cases where anycast does not direct the clients ($\sim$20%) to the best optimal front-end, a simple history-based prediction helps DNS redirection for underserved clients.

Due to the drastic growth in the number and the size of web objects on a web page, data compression has become prevalent. Content compression has shown promising results for improving the web performance especially on the mobile web [54]. Data compression and encoding techniques such as GZip compression for the web [55] and WebP for image compression [56] have been introduced to reduce the data consumption and the latency. However, these techniques are not adopted very well by the content providers. For instance, Agababov *et al.* showed that 42% of the web contents that would benefit from GZip compression are still uncompressed, and only 0.9% of the images in the web are encoded with the WebP format [54].

Agababov *et al.* presented a compression system for proxied web content [54]. They showed that their system compresses 58% (on average) of the web content when accessed using Chrome for Android and iOS devices. The authors noted that, although compression improves latency in general, proxying may add extra latency. Hence, although proxied compression plays a significant role in data reduction, its impact on the latency improvement varies on the metrics of interest and types of web page. For instance, it reduces the PLT for large web pages but increases it for smaller web pages. Singh *et al.* [57] proposed a network-aware compression method for mobile webpages. They showed that their approach reduces the PLT for mobile clients by $\sim$35-42%. In addition to compression, other content provision mechanisms such as content prioritization [58] have been used to improve mobile web performance.

2.2.3 Protocol Evolution

HTTP and TCP are protocols that have been used as the building blocks of the web. HTTP, the application level protocol has evolved [59] from the first 'one-line protocol' – `HTTP/0.9` – to the 'versatile protocol' – `HTTP/1.0` – and sooner to the 'standardized protocol' – `HTTP/1.1` [60]. By adding various extensions and feature improvements, the standardized version HTTP/1.1 has been used for over 15 years. Important extensions to HTTP/1.1 include state management [61], client initiated content-encoding [62], HTTP over TLS [63], conditional requests [64], range requests [65], caching [66], and authentication [67]. Despite these and other extensions, the web still suffers from latency.

However, by design HTTP/1.0 uses a single connection per request, client initiated requests, uncompressed requests and response headers, redundant headers, and optional data compression. For the last 15 years, HTTP/1.1 has used multiple TCP connections to process multiple requests simultaneously. However, it consumes too much network resources and leads TCP congestion [68]. This has an impact on the network performance. There is also a limit to the maximum number of parallel TCP connections that a browser should create (typically browsers open 6 to 8 parallel connections). So if there is a need to open other multiple connections, the browser has to wait for the previous connections to close. Hence, HTTP/1.1 added extensions such as persistent connections and request pipelining [68], but it still suffers from head-of-line blocking. Moreover, HTTP/1 request header fields are verbose and repetitive which leads to the TCP initial congestion window filling up quickly [69]. The request and response headers are also uncompressed which adds extra serialization latency to send requests. Another problem with HTTP/1 is that all requests are initiated exclusively by the client. Even if the server knows that the client needs a particular resource, it has no way to inform the client and has to wait until the client requests the resource. In addition to that, data compression is optional in HTTP/1 and inefficiently used.

To overcome the inefficiencies of the HTTP/1 and reduce the web latency, there have been a number of proposals for the transport and session layer including multiplexing and simple multiplexing (SMUX) [70]. To reduce the latency, there have been also various extensions to TCP. The extensions include landing reordering [71], packet losses [72], increasing the initial congestion window of TCP [73], TCP Fast Open [74], and multipath

TCP [75]. While the proposals and extensions at the transport layer have brought improvements in latency, there is also a need for encrypted traffic which adds extra overheads. As such, further optimizations are required to reduce the latency overhead due to encryption in the transport layer. For instance, the recently standardized transport layer security protocol – TLS 1.3 – [76] is designed to reduce the latency due to encryption. While these optimizations are promising, their deployment are challenged by middleboxes and legacy systems [77]. Although the aforementioned proposals reduce web latency, there are still other problems with HTTP/1 that need to be fixed such as header compression, stream prioritization, and exclusively client initiated request handling.

In 2010, Google proposed SPDY (pronounced as 'SPeeDY') [78], a web protocol that runs on top of TCP and augments the HTTP/1 features to achieve a better performance. The main goals of SPDY were to reduce latency and improve the page load time, reduce deployment complexity, and avoid content changes because of the changes in the protocol [79]. More specifically SPDY had the objective to enable concurrent HTTP requests over a single TCP connection (multiplexing), reduce the bandwidth by compressing important and removing unnecessary HTTP headers (header compression), enabling server initiated content pushing when ever possible (server push) or providing suggestions to the client to request a specific resource (server hint), and better security using Secure Sockets Layer (SSL) [79]. SPDY was first enabled in the Google Chrome browser and adopted by big CDN providers such as Google and Cloudflare.

SPDY brought the motivation to explore further for an alternative web protocol that would alleviate the inefficiencies of HTTP/1. In 2015, the extended version of HTTP was standardized and named as HTTP/2 [69]. The main goals of HTTP/2 include multiplexing and concurrency, stream prioritization, header compression [80], and server push. The adoption of HTTP/2 is steadily increasing. A study by Zimmermann *et al.* [81] showed that, in 2017, about 5.38M domains under the `.com`, `.net`, and `.org` domain spaces supported HTTP/2. The study also revealed that, although millions of domains support HTTP/2, only 595 domains use the server push feature.

However, even though HTTP/2 was designed to overcome the limitations of its predecessor, it does not completely alleviate all the inefficiencies of HTTP. While the standardization of HTTP/2 was going on in 2012, Google also proposed "Quick UDP Internet Connections" (QUIC), a transport protocol that aims at reducing the latency compared to that of TCP [82]. It is

implemented on top of UDP and has a similar objective to the existing web stack (TCP+TLS+HTTP/2). The key advances of QUIC over the existing TCP+TLS+HTTP/2 web stacks include low latency, improved congestion control, multiplexing without head-of-line blocking, packet pacing, and no encryption overhead [83]. Google has enabled QUIC for its services, and the Internet Engineering Task Force (IETF) QUIC working group is also working on standardization. In 2018, the working group named HTTP/2 over QUIC as HTTP/3 [84]. While there are a number of experimental implementations of IETF QUIC [85], to the best of the author's knowledge, it has been deployed in a production-scale only at the Facebook data center [86]. The Google QUIC implementation has been deployed mostly on Google services. A live measurement dashboard [87] by Hohlfeld *et al.* shows that as of April 2020, there are ∼1.3M IETF QUIC capable hosts in the IPv4 address space.

2.3 Web Performance

Web performance is paramount for the end user experience. It has a direct impact on the content providers' and ISPs' business revenue. A few milliseconds of delay in the download of a web page could result in a significant loss in business revenue for the service provider due to customer churn [88]. For instance, a 0.5 second delay in Google search page result drops Google's traffic by 20%, and every 100 ms latency decreases Amazon's sales by 1% [89]. Over the last years, web performance has been well studied and there have been also various proposals to improve the performance. In this section, we first present the factors that affect web performance, and then we discuss optimization techniques.

2.3.1 Factors Affecting Web Performance

A number of factors such as the underlying transport, the client device computational power, web page complexity, latency due to the load at the web server, and the browser implementation affect web performance [90].

Transport protocol. As mentioned in Section 2.2, TCP has been the de-facto transport protocol for the web. Since TCP was designed as a general Internet transport protocol, it has performance related inefficiencies in delivering web pages. Wolsing *et al.* [83] performed a performance comparisons study on the state of the art web stack (TCP+TLS+HTTP/2)

and QUIC. The authors also make a comparison by tuning TCP variants. They showed that QUIC outperforms the tuned TCP variants. Rüth *et al.* [91] studied the how users perceive the impact of Quick UDP Internet Connections (QUIC). The authors showed that users did perceive QUIC as a fast protocol in a side-by-side comparison with the normal and even tuned TCP. When the users have a fast network, they do not have any preference in the protocol. However, in lossy and slow networks, the advanced protocol design of QUIC brings an advantage by improving the web loading process. This shows that QUIC has the potential to improve bad browsing experiences on lossy and slow networks. The performance advantage in QUIC comes from the single RTT during the connection establishment and in lossy networks due to its ability to overcome the head-of-line blocking problem.

Name resolution and content encryption. Almost all web pages are identified by human readable domain names. Before the web pages are fetched, these names need to be mapped with a corresponding Internet Protocol (IP) address of the server that hosts the content. The process of converting a domain name to an IP address is called DNS resolution. This process adds an extra overhead to the download performance of the web page. A study by Walelgne *et al.* [92] shows that the DNS lookup time for different web pages varies significantly. This is due to the presence of DNS cache entries at the ISPs. The lack of good DNS servers and caching infrastructure is one of the main performance bottlenecks [93].

Until recently, the DNS resolution queries were performed on top of UDP and the queries were not encrypted. To mitigate the privacy issues of sending DNS queries in the clear, recently, researchers proposed encrypted DNS queries on top of TLS and HTTPS. Hounsel *et al.* studied the performance difference between the conventional DNS (Do53), DNS over TLS (DoT) and DNS over HTTPS (DoH) [94]. The authors showed that although the response times of DoT and DoH were higher than Do53, in a normal network conditions DoT performed better than the other two in terms of PLT. However, when the network condition degrades, the web pages load more quickly in Do53.

Moreover, users have become more conscious of security over the past years, which has led to an increase in encrypted HTTP traffic over the Internet. Since Transport Layer Security (TLS) needs more RTTs than a normal TCP connection, the security feature of HTTPS comes with a cost on infrastructure, communication latency, data usage and energy

consumption. As a result, the use of HTTPS increases the web page load time for websites [95].

Client device computational power. Limited computational power causes a delay in processing and rendering of the web pages. Nejati *et al.* [90] did a comparison of page loading activities and bottlenecks when loading a web page in mobile and non-mobile browsers. The authors showed that the main bottlenecks in mobile and non-mobile browsers were the computational activities and the network activities, respectively. They also observed that the composition of the critical paths was different when loading a web page in mobile and non-mobile browsers. Vesuna *et al.* [96] showed that CPU speed is the bottleneck that prevents mobile devices from taking the advantage of web caching.

The capability of the end-user device has an impact on the browsing performance. A study by Ahmad *et al.* [97] showed that device-level bottlenecks such as limited processing power were among the reasons for poor web performance in developing regions. Dasari *et al.* [98] analyzed the impact of device performance on the mobile internet QoE. The authors showed that web browsing was more sensitive for low-end hardware than video streaming. This is because video applications use a specialized hardware for decoding the video – which is also available in low-end devices – and on multi-core mobile devices parallel operations are used for post-processing (*e.g.,* muxing and demuxing video/audio). The authors also showed that web browsing is affected by the clock speed, because usually it uses no more than two cores simultaneously.

Availability of non-cacheable contents in critical path. Vesuna *et al.* [96] investigated the effect of caching on mobile web performance. They concluded that caching alone does not improve the web performance much. The authors showed that the weak performance was due to the fact that the objects in the critical path are often not cached and the mobile devices have limited computational power.

Webpage complexity. Web pages consist of tens to hundreds of web objects that can be HTML, JavaScript, CSS, images, web fonts, multimedia content such as video and Flash, etc. number and Although the complexity in terms of the number and the size of the objects does not directly translate into the performance of a web page [99], processing the objects and rendering to the browser brings a performance overhead. The impact of the complexity of web pages on the loading performance is higher for mobile web pages [58]. For instance, processing JavaScript in low-end mobile de-

vices has a worse performance than high-end devices [100]. Synchronous JavaScript calls play a key role in the page loading latency by blocking the HTML parsing [101]. Wang *et al.* [102] also showed that parsing-blocking CSS and JavaScript files slowed down the web page load time by 20%.

Browser rendering engine. Different browsers use different architectures (*e.g.,* uni-processor, multiprocessor) and rendering engines. That is, Chromium-based browsers such as Google Chrome and Brave use the Blink engine, Firefox uses the Gecko engine and Safari uses the WebKit rendering engine to interpret HTML documents and transform web page elements to visual representations on user devices. The scripting engines used to interpret JavaScript vary between browsers. For instance, Chromium-based browsers use V8, Firefox uses SpiderMonkey, Microsoft Edge uses Chakra, and Safari uses JavaScriptCore for parsing and executing JavaScript [103]. Consequently, different browsers have different dependency policies when loading objects [101], JavaScript execution, and CSS evaluation. For instance, JavaScript is blocking for the `onLoad` event in Firefox but non-blocking for Internet Explorer [104]. These variations in the browser implementations result performance differences.

Wijnants *et al.* [105] studied the effect of user-agent implementations and the HTTP/2 prioritization approaches on web performance. They showed that browsers use different prioritization approaches – naive (Safari, Internet Explorer, and Edge) and complex (Chrome and Firefox). The authors showed that complex prioritization approaches yield better performance. Instead, the naive approaches can lead to 25% slower PLT. They also showed that prioritization matters most on heavy-weighted web pages.

Third-party contents. In addition to the main content, web pages contain extra contents (mostly JavaScript and CSS files) owned by other content providers referred to as 'third-parties'. The web pages use these third-party contents for various purposes such as analytics, advertisements, service personalization based on user preference, and to make the web page more interactive and aesthetically appealing [106]. However, these contents may not be optimized for a specific website, and bring additional performance overheads. Content providers make a contract with CDNs to host and deliver (static) web contents in close proximity to the end user. Nonetheless, the CDNs do not have control over the contents owned by a third-party and cannot optimize them for performance improvements. A study by Goel *et al.* shows that third-party resources in the webpage's critical path contribute up to 50% of the PLT. The authors also showed

that using URL rewriting techniques to serve the third-party resources from the primary infrastructure would optimize the PLT up to 25%.

Moreover, to reduce the performance impact and security concerns [107] due to third-party contents, content blockers (*e.g.,* ad blockers to restrict advertisements) have been widely used. For example, the use of ad blockers reduces the number of objects. A study by Newman *et al.* [108] shows that ad blockers reduce up to 15% (median case) of the loaded objects which could yield up to a 12.5% improvement in PLT. Despite the improvement in PLT, the use of ad blockers slows down the initial responsiveness of the website. It increases the time to first paint up to 19%. The authors showed that users prefer a faster first paint over faster page load times.

2.3.2 Web Performance Optimization Techniques

To deal with performance issues due to the aforementioned factors and improve the web performance, several techniques have been proposed. The proposals include content caching, content prioritization, and new transport and application protocols.

Caching. One way of improving the web performance is to reduce the latency of delivering the web content from the server to the client. That is, shorter latency in getting the web content is always preferred by the end-user. In order to reduce the latency and improve the performance, content caching has been used by content providers. For instance, the introduction of CDN and client-side caching [109, 110] are some of the techniques introduced to place part of the web page content closer to the end-user or in the client device itself. Moreover, to improve the performance of the mobile web, various caching techniques including Google Accelerated Mobile Page (AMP) [111] have been proposed and CDNs continue to adopt it. Proxying and content prefetching have been also used to reduce web latency [54]. In addition to content caching, DNS caching and HTTP redirection caching also help improve web performance. For instance, the authors in [93] showed that simple DNS caching and redirection caching can yield a substantial improvement in the user-perceived performance.

Using reader mode. Browsers provide a feature called 'reader mode' that eliminate less relevant elements of the web page such as advertisement images and videos. That is, browsers remove unrelated contents and render only the main contents of the web page such as HTML and CSS files. A study by Ghasemisharif *et al.* [112] shows that using the reader mode gives on average up to 27× (in the median 15×) page load time improvement,

on average up to 84× (in the median 24×) bandwidth reduction, and on average up to 2.4× (in the median case 2.1×) memory reduction. Moreover, the authors showed a significant number of websites can be converted into a reader mode. The reader mode is mostly suitable for web pages that have mainly static contents such as news pages, blog posts, etc. Note that some web pages that have interactivity features such as Gmail and Google maps cannot be converted to a reader mode

Better protocols. As we discussed in section 2.2, various application and transport protocols have been proposed aiming at improving the performance of the web. HTTP/2 [69], and QUIC [82] are amongst the recent protocols developed to improve the web performance. Rosen *et al.* [113] studied the benefits of using HTTP/2's server push [81] feature in mobile devices. They showed that server push had better benefits to high loss rates and high latencies, but had a limited benefit for high-speed connections. Server push reduces energy consumption, which is one of the challenges in provisioning better web performance and QoE for mobile devices. The authors [114] studied if the current web is ready for HTTP/2 server push. They found that the web may be technically ready to use server push, but the feature cannot be used easily. That is, when and how to use the server push feature is specific to websites. They performed a measurement study of how the server push feature impacted the performance of websites. They showed that with the general guideline to push all embedded objects the performance improved on some websites, but also decreased for others .

Erman *et al.* [115] performed a measurement study to understand the benefits of using SPDY [78] over cellular networks. The authors showed that SPDY does not outperform HTTP in cellular networks. The reason for this is that SPDY uses a single TCP connection and this problem gets worse when the device goes through a radio access technology change.

Content prioritization. Prioritizing the delivery of contents in the critical path of the web page yields a performance improvement. Different techniques can be used for the prioritization such as rewriting the webpage, prioritizing the contents in the above-the-fold area, or dynamic prioritization [58]. The authors in [58] developed `Klotski` that dynamically prioritizes web objects. They showed that using their prioritization system helps improve up to 60% of the user perceived utility. Web protocols such as HTTP/2 have resource prioritization mechanism as a part of the protocol specification. As mentioned before, a study by Wijnants *et al.* [105] revealed that the performance benefits of HTTP/2's resource prioritization

feature depend on the browser implementation. For instance, in browsers that use complex implementation (*e.g.,* Chrome), HTTP/2's resource prioritization yields better performance and reduces the visual load time in naive implementation (*e.g.,* Internet Explorer). This is because browsers use different heuristics to prioritize requests.

Wang *et al.* [102] designed a system that prioritizes resources that are necessary during the initial page load. The system gives lower priority to objects that are required after the page loading process. Unlike the other prioritization methods [58, 105], the authors argue that the prioritization should consider both network transfer and computation at a fine granularity level. They showed that their prioritization scheme improved the PLT by 50% to 60% on 1GHz CPU phones.

Compression and lazy loading. Content compression has been used to reduce the data consumption and latency. Compression techniques like GZip [55] have been used to compress HTML, CSS and JavaScript files. Recently, Google also proposed a new image format – WebP [56] – primarily for web pages. It can compress images by up to 30% with the same quality as JPEG. In addition to compressing web objects, non-critical objects can be loaded at the time they are needed instead of upon page load. This technique is called lazy-loading [116]. For example, images and videos that are located below the current view port of the browser are not critical elements for the web page loading, and they can be loaded when the user wants to see them.

Optimizing JavaScript loading, parsing and execution. The use of JavaScript is becoming prevalent in modern web applications. Usually a single file contains multiple JavaScript functions. Traditionally, a browser's JavaScript parser needs to parse the whole JavaScript file before executing a function. This parsing and executing of JavaScript files is an expensive computation, which slows down the web page loading process. However, modern parsing engines allow lazy parsing. That is, instead of parsing the entire code, they parse those functions and the global code that need to be executed. Various techniques such as unused code elimination [117], reducing payload with code splitting [118], concurrent parsing [119], and compiled code reusing [119] have been proposed to optimize the JavaScript loading, parsing and execution process. A study by Park *et al.* [119] shows that concurrent parsing improves the web application loading performance by up to 32.7% and by 18.2% on average.

2.4 Web Quality of Experience (QoE)

According to the International Telecommunication Union (ITU) QoE is defined as "the overall acceptability of an application or service as perceived subjectively by the end-user" [120]. A Qualinet white paper [121] defines QoE as "the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state." These definitions consider that the QoE in communication systems and applications depends on the device, the network, the user mood and expectations, the context and purpose of use. Starting from these definitions, web QoE is the acceptability of a service delivered through a web page.

2.4.1 Measuring Web QoE

The emergence of various web applications such as such e-commerce, social media, news and entertainment websites have changed the primary design and use of the web. For different kinds of websites, the end-users have different requirements. The key parameters for the end-users to evaluate the QoE of web applications are the waiting time to get the requested content [122] and the usability of the service [123]. The shorter the waiting time to get the content of the webpage, the more the users are satisfied with the service [122]. Researchers have proposed three time limits for subjective response times for interactive systems such as the web [124].

- 0.1 second is a limit where the user feels that the system is responding instantaneously.
- 1 second is the limit that the user considers the system to continue to work, even if the user notice delays in the response.
- 10 seconds is the limit considered when the user potentially stops paying attention to the system and abandons using the system.

Nevertheless, since different factors affect the users' browsing QoE, user satisfaction and expectations are not directly linked to these limits. Factors include the application, the previous experience (which defines user expectations), the context of use, network and device factors, and the browsing purpose [122, 125]. As a result, measuring the QoE for web applications has been challenging for various reasons. On the one hand, there is a need for unified metrics that enable the measurement of QoE on a large scale.

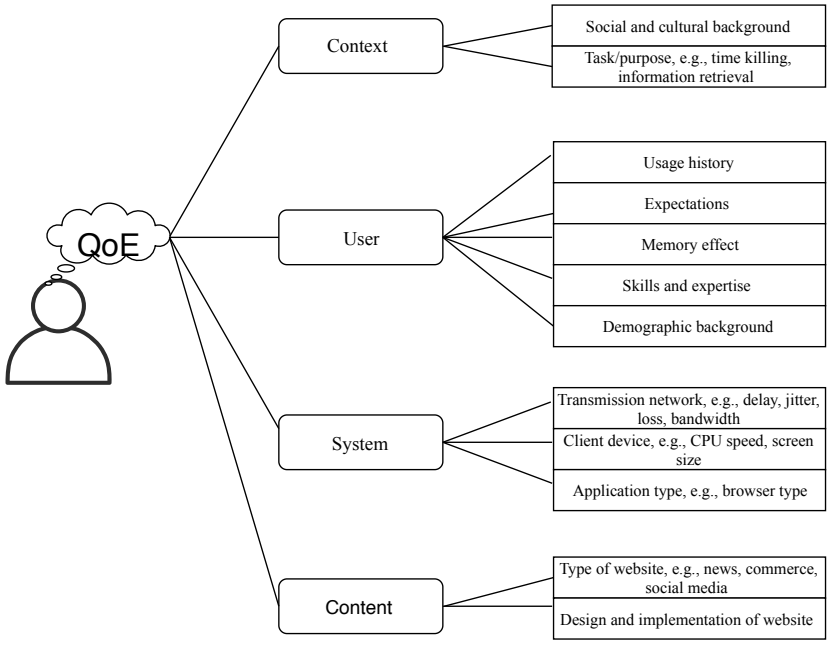

Figure 2.3. QoE influencing factors for web application. *Source: adopted from [128]*

On the other hand, the web applications are different in design, type and purpose which makes it difficult to have a unified model for predicting the web QoE from measurements at the application and network level.

2.4.2 Factors Affecting Web QoE

The web ecosystem is a complex system which involves multiple stakeholders who play a role in the delivery, processing and rendering the content. As such, a number of factors influence the overall web browsing QoE. The factors are either technical factors related to the accessing system and the content or social and psychological phenomena that are related to the user [126, 127]. Figure 2.3 summarizes the influencing factors as outlined in the web QoE definition. The curved boxes on the left show the categories and the boxes on the right show some examples in each category.

Context. Context is anything that can be used to specify or clarify the meaning or situation of a particular event. The context-related factors are those that describe the overall condition while the user is performing the browsing task. The physical context, the social and cultural context, the purpose of the task that the user is performing, the time pressure, and the interruption cost are contextual factors that influence the web QoE [129]. For instance, users who surf the web for leisure do not have

the same browsing experience as those doing so for information retrieval. These kinds of factors are hard to measure directly.

User. There are also influencing factors that directly emanate from the user. Factors such as the users' skills and expertise, memory effect, previous user history, expectations, needs, goals, personality traits, preferences and sentiments, and the demographic background are user related influencing factors that affect the web QoE [129]. For instance, a user surfing the web in developing regions may not expect the same experience as in the developed regions. A similar delay experienced in the developed regions that would lead the user to abandon using the website may not impair the QoE of a user in the developing regions. According to Hoßfeld *et al.* [130], the user time-dynamics and the internal state of the user are key factors that influence the web QoE, and these kinds of factors need to be taken into consideration in modelling the QoE.

System. Factors that are related to the access system such as the network and end-user device also affect the web QoE. Factors including delay, jitter, loss in the transmission network, the performance of client devices such as the screen size, the CPU speed and the available memory, and the application, *e.g.,* the browser type are system related confounding factors that influence the user browsing experience. For example, the user may perceive different levels of quality in accessing the same website from the same device and transmission but with different browser types [131, 132].

The system-related factors can be either provisioning- or delivery-related factors, which have been shown to have a different impact on the QoE [133]. While provisioning-related factors have a positive impact on the QoE, delivery-related factors adversely affect the QoE. Influencing factors related to the system are the least complicated to measure than the other factors. As such, these factors have been considered as an input for various QoE prediction models [129].

Content. The type, design and implementation of the website have also a significant impact on the user browsing experience. For instance, the location of the JavaScript on the web page may block the download of web objects which has an adverse effect on the user experience. The design of the website determines the usability of the website, which is one of the influencing factors for the end-user experience [123].

2.4.3 Metrics for Web QoE Evaluation

Perceived QoE is subjective in nature, hence, measuring it requires direct end-user involvement. For networked applications such as the web, subjective user opinion has been widely used to quantify the perceived QoE. Mean Opinion Score (MOS) [8] is a commonly used user opinion-based metric to quantify the end-user experience. It is expressed on an ACR scale usually in a range from 1 to 5, where 1 represents bad quality (high impairment) and 5 represents an excellent quality (low impairment). The MOS is calculated by taking the average of the user opinions. However, taking the mean value of the opinion results in loosing important information on the user diversity [9]. This is among the shortcomings of using MOS to represent the QoE. To overcome the limitation of MOS, other user opinion-based metrics such as the standard deviation of an opinion score [134], Θ-Acceptability, and the ratio of (un-)satisfied users are also proposed [9]. Although these metrics have not been widely used in practice, their goal is to provide a comprehensive view of how the quality is perceived by the users.

In general, using user opinion score-based metrics becomes impractical and prohibitively expensive as it is difficult to collect the user ratings on a large scale. User opinion score-based metrics are used as a benchmark for mapping objective QoS metrics to QoE using either mathematically formulated expert models or data-driven models, *e.g.,* using machine learning techniques.

To overcome the limitations of user opinion score-based metrics, researchers have proposed objective metrics in order to objectively estimate the perceived quality from measurements performed in the application and the system itself. For objectively quantifying the web performance and browsing QoE, the waiting time for getting the content of the website is a commonly used metric. Content providers and ISPs use different points of interest in the process of the web browsing session to quantify the waiting time. For instance, ISPs may want to understand the time elapsed when a customer connects to the web server, or the time up to the first byte of the response, or the time required to download the whole content of a web page.

Some of the metrics in this class like TTFB are easy to capture, for instance from the network trace. While some others including Time to Interactive (TTI) and Above The Fold (ATF) time need browser instrumen-

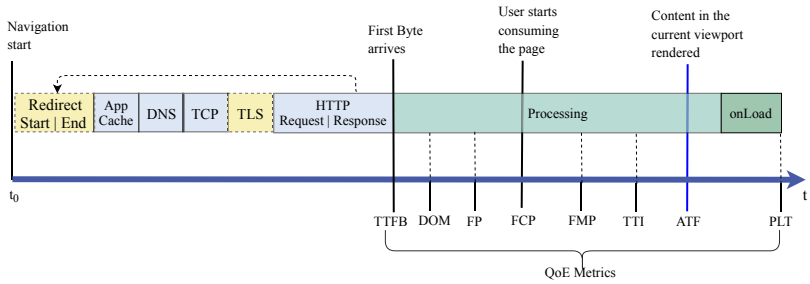

Figure 2.4. Browser events and time-instant web QoE metrics when downloading and rendering a web page. Events in doted boxes are optional, which may not exist fow all browsing phenomena.

tation or other processing tools to capture them.

The objective metrics are either time-instant or time-integral measures [7].

Time-instant Metrics

Time-instant metrics are proposed to measure the end-user experience directly from the browser or the communication network without requiring user interaction. The time-instant metrics are computed by observing the occurrence of a particular event during the web page download process. Most of these metrics are measured at the application level, for example, using the browser navigation timing information. Figure 2.4 shows the events in the browser such as DNS lookup, TCP and TLS handshake, the time to receive the first byte of the payload (TTFB), and the timeline of the time-instant QoE metrics in the web page download process. Different short web flow latency metrics including HTTP redirect, DNS lookup time, TCP connection time, and TLS handshake time are implicitly taken into consideration when calculating the time-instance QoE metrics. The metrics are computed by taking the difference between the occurrence of the event to the beginning of the navigation (t_0 in Figure 2.4).

Time-instant metrics can be categorized as visual and non-visual metrics [135]. Visual metrics are metrics that measure the occurrence of an event that shows a visual change on the browser screen.The time to first paint (FP), first contentful paint (FCP), first meaningful paint (FMP), TTI, and ATF time are visual metrics. In contrast, non-visual metrics are metrics that measure the occurrence of an event which does not have a visual change effect on the browser screen. TTFB, Document Object Model (DOM) time and Page Load Time (PLT) are non-visual metrics. The details of each time-instant QoE metrics are described below.

Time To First Byte (TTFB). This is the time from the beginning of the

navigation (t_0) to the arrival of the first byte of the payload of the web page.

Document Object Model (DOM) time. This represents the time at which the critical elements of the web page are downloaded and the DOM tree is parsed and the browser is ready to download additional necessary objects that are necessary to build the web page.

Time to first paint (FP). This is – also known as first visual change – the time at which the browser starts painting or rendering the first pixels of the web page. The pixel can be anything that is visually different from prior to the navigation. This shows that something is happening in the browser screen.

Time to first contentful paint (FCP). this is the time at which the browser paints or renders the actual web content, such as text, image (including background images), and non-white canvas. This indicates that something useful is rendered on the screen and the user starts consuming the web page.

Time to first meaningful paint (FMP). This is the time at which the browser renders the primary meaningful content of the web page. The primary content varies between web pages. For example, for a news web page the headline and the text in the above-the-fold area (including the required web fonts) are the primary contents.

Time to Interactive (TTI). This is the time at which the web page is rendered and the browser is able to respond to the user interaction (*e.g.,* clicking a link, tap on a button). This identifies when the initial JavaScript in the web page is loaded and the main thread is idle. A variant of TTI is time to first input delay (FID) (not shown in Fig. 2.4). Unlike TTI, FID measures the time from when the user first interacts (*e.g.,* clicking on a hyperlink) with the web page up to when the browser starts responding for the user action.

Above The Fold (ATF) Time. This is the time at which the web page completes downloading and rendering the contents within the above-the-fold area (*i.e.,* within the current viewport of the browser). The rest of the contents in the web page may still continue to download after the content in the above-the-fold area are displayed. However, they do not change the look of the web page. This metric is also known as the rendering time [136] or last visual change.

Page Load Time (PLT). This is the time from the start of the navigation to the time at which the browser finishes downloading all the objects on

the web page such as HTML files, scripts, images, and style sheets and the `onLoad` event fires. Sometimes the browser may keep the TCP connection open and the `onLoad` event may fire before all the objects are downloaded. That is, objects may still be downloading asynchronously, even though the `onLoad` even has already fired. This metric has been commonly used as a de-facto metric for evaluating web QoE.

Note that the implementations of the timing events are browser dependent. Different browsers use different rendering engines (*e.g.,* Chromium-based browsers like Google Chrome and Brave use the Blink engine, Firefox uses the Gecko engine and Safari uses the WebKit engine) to interpret HTML documents and transform webpage elements to visual representation on the user devices. Consequently, the timing of events such as `onLoad` may vary between different browsers for the same website.

Time-integral Metrics

Time-integral metrics are other kinds of objective metrics which aim to objectively estimate the perceived QoE from measurements performed at the application or system level. Time-integral metrics are measured by integrating the time to the events within the web page download progress [7]. The metrics are generalized as follows:

$$X = \int_0^{t_{end}} (1 - x(t))dt \tag{2.1}$$

Where X is the value of the metric, t_{end} is the time to the last event in the web page waterfall, and $x(t) \in [0, 1]$ is the time evolution of the progress in reaching to the event. The following are the time-integral metrics proposed to approximate the perceived browsing quality.

Speed Index. Proposed by Google [137], this represents the time until the visible portion of the web page is rendered. It is computed by taking a series of screenshots of the web page download progress and post-processing the captured frames [11]. One of the challenges for calculating the Speed Index is that, it is computationally intensive which makes it difficult compute for large sets of websites on a large scale. Gao *et al.* proposed the PerceptualSpeedIndex as a complementary metric to the Speed Index. While the Speed Index focuses on addressing how the web page content in the ATF area loads quickly, the PerceptualSpeedIndex focuses on addressing if the web page content in the ATF area loads without a visually perceivable

jitter [135]. Hoßfeld *et al.* [12] gave a theoretical formulation of the Speed Index and evaluated the interdependency between the Speed Index and MOS. They showed that ATF-based metrics are more appropriate for web QoE models than the pure PLT.

Byte / Object Index. Motivated by Google's proposal, Bocchi et al. proposed the Byte Index and Object Index. Byte Index and Object Index are metrics that estimate the time to byte level and object level completion of a web page, respectively [7].

Ready Index. Netravali et al. [138] proposed the Ready Index, which measures the page load time by considering the functionality and web object readiness. The Ready Index is the time the web objects in the above-fold-area area are rendered and became ready, e.g., when the search text area and buttons in a shopping website are ready to take the user input.

Time-integral metrics can have multiple cut off points for integration [11]. For instance, the ATF time may be cut off point for the Speed Index calculation. The ATF time, PLT, FCP or FMP can be a cut off point for computing the Byte Index and Object Index of a web page.

2.4.4 Web QoE Models

Researchers have proposed various models that map the objective measurements to the perceived QoE expressed in terms of MOS [12]. The models can be expert models that express MOS as a function $f(\cdot)$ of the waiting time t, *e.g.,* PLT [131]. There are also data-driven models that use different QoS metrics as a feature and apply machine learning techniques to learn the mapping function from the data [139, 11, 135, 140]. The commonly known expert models are logarithmic and exponential models.

Logarithmic Models

The QoE models of this class follow the Weber-Fetchner law hypothesis applied to the waiting time [141], and are shortly referred as WQL models. WQL models assume that the 'W'aiting time and the 'Q'oE have a 'L'ogarithmic relationship [12]. This is expressed as follows:

$$QoE(t) = \alpha log(t) + \gamma \tag{2.2}$$

where, α and γ are parameters and t is the waiting time.

ITU-T G.1030 [142] follows the WQL hypothesis and proposes a recom-

mendation for modeling the web QoE in terms of the session time. The recommendation is based on a simple image retrieval task from a webpage. The model follows a context-aware approach by considering three different time scales – 5s, 15s, and 60s – corresponding to fast, medium, and slow network contexts, respectively.

Exponential Model

These models stipulate the exponential interdependency of the quality of experience and quality of service (IQX), which assumes that for a given stimulus t, the changes in the perceived QoE depend on the current level of QoE. That is, if the QoE is high, a small degradation in the underlying QoS metrics will have high impact on the resulting QoE. If the QoE is already bad, a small degradation will not have a significant impact on the resulting QoE. These models are expressed as follows:

$$QoE(t) = \alpha e^{-\beta t} + \gamma \tag{2.3}$$

where, α, β and γ are parameters and t is the waiting time.

Machine Learning-based Models

Researchers have tried to learn the QoE model from the data using the various network characteristics and QoS metrics as features. Balachandran *et al.* [139] designed a machine learning model to predict the impact of various network characteristics such as signal strength on the mobile web QoE. The authors showed that the web QoE is sensitive to change in inter-radio-access-technology (IRAT). Hora *et al.* [11] evaluated different machine learning techniques to learn the regression function from different QoS and objective QoE metrics such as the PLT and ATF time to MOS. The authors evaluated three machine learning algorithms: Support Vector Regression (SVR), Classification And Regression Tree (CART) and AdBoost with CART. They showed that SVR outperformed both algorithms. The authors in [135] also built a predictive machine learning-based perception model using different metrics as features. They tried Random Forest and Gradient Boosting algorithms. The features include PLT and *SpeedIndex* with different cut off points such as the time to click and visual complete. The authors showed that taking three of the metrics (*i.e.,* time to FCP, *SpeedIndex* and *PerceptualSpeedIndex* with the time to click as a cut off point) yields the same level of accuracy as all metrics combined.

Trevisan *et al.* [140] proposed a system called PAIN (PAssive INdicator) to automatically monitor the web performance from passive measurement by leveraging flow-level DNS measurements. PAIN uses unsupervised learning techniques to measure the web performance by mapping the DNS requests issued by browsers to render the webpage. The authors showed the metrics used in PAIN are strongly correlated with the web QoE metrics such as PLT and the Speed Index.

2.5 Methods and Tools for Web Performance & QoE Measurement

In this section, we discuss the recent methods and tools that have been developed to measure and evaluate the web performance and network properties. We present tools that can be used for server-side benchmarking as well as client-side performance monitoring. Most of these tools are not suitable for large scale deployment. For instance, they require user interaction to run the experiments. They also do not consider the most critical metrics for approximating the user experience. The measurement system and tools we developed close these gaps by better approximating the user browsing experience and do not require user interaction to execute the experiment on a large scale.

2.5.1 APIs and Standardization Efforts

The W3C Web Performance working group [143] standardizes the web performance measurements and APIs in web browsers. The group has specified several web performance measurement APIs. Among those, the Navigation Timing, the Resource Timing, and the User Timing APIs help measure the performance of a website on a real world Internet connection [144]. Additionally, the Page Visibility [145], Efficient Script Yielding [146] and Display Painting Notification APIs provide basic information about the rendering state of the web page and help developers write resource-efficient (CPU and power) web applications. For instance, the Page Visibility API enables a developer to determine the current visibility of the page. We use these APIs when we develop our measurement tools.

2.5.2 Client-side Performance Testing and Monitoring tools

WebPageTest. *WebPageTest* [147] is an online tool that enables Internet users to examine a complete web page loading time with different test

browsers, from multiple locations with different network characteristics. It measures the page loading time by dividing the page load session into the DNS lookup time, TCP connection establishment time, SSL handshake time, HTTP response time, etc. The tool also measures the resources consumed during the measurement. However, WebPageTest is an interactive tool that is not suitable for large-scale measurements.

Sitespeed.io. *Sitespeed.io* [148] is a web performance testing tool that uses a set of open source tools and reports the metrics collected by those tools. The underlying tools that *sitespeed.io* uses to measure the performance of the web are:

- *The Coach* is a modern version of *YSlow* [149] that helps find a performance problem in a website.
- *Browsertime* collects the metrics by querying the timing data directly from the browser. It runs custom JavaScript to get the statistics for each run, and generates the HTTP Archive (HAR) files. Furthermore, it records a video of the screen and analyzes the result to get the first and last visual changes, the speed index, and 85% visual complete time.
- *PageXray* converts the HAR file into the JSON format that tells more about the page.

YSlow. *YSlow* [149] analyses a web page and gives suggestions why the web page is slow and a way to improve the performance based on 23 rules of Yahoo's 34 rules [150] for high performance website. It grades web pages based on predefined rules or user-defined rule set, summarizes the page components, displays statistics and provides suggestions on how to improve the web page performance.

Pingdom. *Pingdom* [151] is an online web analysis tool that finds bottlenecks along with the website's uptime, performance and interactions for a better user experience. Pingdom helps examine the web page characteristics such as file size, type, details about each element, and other performance-related statistics from different locations. The test is done using a browser to represent the end user experience.

GTMetrix. *GTMetrix* [152] tests and monitors a website performance. By using the PageSpeed and YSlow tests it grades and provides reports to improve the performance in order to reduce the web downloading time. It also gives detailed information about the page such as the page loading time, the web page size and number of requests.

Google PageSpeed Insights. *Google PageSpeed Insights* [153] is a tool that analyses the content and speed of a webpage and it gives recommendations to make the website faster both on mobile and desktop platforms.

Pwmetrics. *Pwmetrics* [154] is a command line performance testing tool which uses the Lighthouse API [155] to get values such as *First Content full Paint, First Meaningful Paint, First Interactive, Perceptual Speed Index, First Visual Changing, Visually Complete 100% and Visually Complete 85%* [156].

PerfWars. *PerfWars* [157] is an online tool that compares the performance of two URLs. It gives detailed information which helps optimize and improve the performance of the website to compete against rivals.

Louis. *Louis* [158] analyses a performance of a website against the performance budget [159]. The tool has an option to set the performance budget for various metrics such as number of requests, image size.

What Does My Site Cost?. *What Does My Site Cost?* [160] is an online tool that tells how much it costs to use a website in a mobile network across the world. The tool provides the information with different options, *e.g.,* the cost with different subscription plans (postpaid and prepaid data plans). The tool calculates the cost based on data from ITU and World Bank.

Uptrends. *Uptrends* [161] is an online tool that offers more than 35 locations for performance testing. The tool gives a performance report in a waterfall breakdown as well as domain groups. In the domain groups report, it categorizes the resources on the website as first party, CDN, social, ads, third party, etc.

Yellow Lab Tools. *Yellow Lab Tools* [162] is a free online tool that tests a web page and detects performance and front-end code quality issues. It loads the web page into PhantomJS [163] for collecting various metrics and statistics about the page. Furthermore, it has options to choose simulated devices (*e.g.,* Desktop, phone and tablet).

WebPerf IO. *WebPerf IO* [164] is a tool that helps to transform the web performance to a business value. It measures the impact of page load time on key business indicators by user segment. The tool is not freely available.

Netalyzr. *Netalyzr* [165] is a web-based network measurement and debugging tool that lets Internet users to evaluate their internet connectivity. It tests different properties of the users' network access, such as the use of IP addresses and address translation, DNS resolver fidelity and security,

TCP/UDP service reachability, etc. *Netalyzer* does not particularly focus on web performance.

Host View. *Host View* [166] is an end host measurement tool for collecting network performance data annotated with the users' perception of the network quality. The tool collects information about the traffic and network performance statistics, application level context, and system performance and environmental data such as the network type.

Mirage. *Mirage* [167] is a headless web client that can be deployed on home router devices to measure the PLT of static contents. The tool downloads the home page of the web site and parses it to determine the static objects that are required to render the page. It separates the page load time into DNS lookup time, TCP connection time, and download time for each object, etc, but does not consider dynamic web contents.

Fathom. *Fathom* [168] is a lightweight browser-based measurement platform that implements a number of primitive measurement APIs. It enables websites or other parties to program network measurement using JavaScript.

Mahimahi. *Mahimahi* [169] is a set of lightweight measurement tools that enables web developers, browser developers and network protocol designers to record HTTP-based applications and replay them over emulated network conditions. It enables users to measure the performance of protocols and other web metrics while recording and replaying.

2.5.3 Measuring Web Dependency

Today's web services run over a complex system that encompasses multiple data centers and content distribution networks. The performance of a web service depends on various factors such as the end-user devices, the network or the hosting infrastructure. Service providers use different optimization techniques to overcome the challenges associated with these factors. Hence, it is important to understand the cost and benefits of these optimization techniques, as well as the impact of the web object interdependencies on the performance.

WebProphet. *WebProphet* [170] predicts the performance of a web service by employing the timing perturbation to extract the web object dependencies. The dependencies are used to predict the performance impact of the object on the user experience. WebProphet has a measurement engine, dependency extractor, and performance predictor.

WProf. *WProf* [101] is a lightweight in-browser profiler that extracts a

dependency graph of activities that make up the page load. It captures the constraints that are raised during network transfer, the page parsing, JavaScript/CSS evaluation and rendering activity in browsers.

2.5.4 Web Performance Optimization and Benchmarking Tools

WebGaze. *WebGaze* [171] is a system that optimizes the user-perceived page load time by prioritizing the web objects that are visually interesting for many of the users using HTTP/2 push feature. The interesting section of the web page is learned from prior measurements collected from a user eye gaze study.

VROOM. *VROOM* [172] is a tool that enables clients to fetch all web objects directly from the domain it hosts and the web server supports the clients in discovering the resources. The authors showed that the tools helps reduce the median PLT of popular news and sport websites by 5 seconds.

Vesper. *Vesper et al.* [138] is a tool that rewrites the page's JavaScript and HTML to automatically detect the page's interactive state. The authors proposed a new metric called the *Ready Index*, which defines the page load time in terms of the interactivity. The tool optimizes the page for the Ready Index and reduces the median time to page interactivity by 29%-32%. The authors also developed Prophecy [173], an acceleration tool to reduce the energy cost, bandwidth consumption and page load time in mobile devices. In Prophecy, the JavaScript heap and DOM tree for a web page is computed at the server side. When a mobile browser requests the web page, the server returns the write log. The tool reduces the median PLT by 53%, energy usage by 36% and bandwidth consumption by 21%. Furthermore, the Apache foundation and others propose various server-side benchmarking and performance testing tools. JMeter [174], Apache Bench [175] from the Apache foundation, Httperf [176], Locust.io [177], Siege [178], and Multi-Mechanize [179] are some of these benchmarking and monitoring tools.

2.6 Summary

This chapter presents the state of the art in web performance and QoE. It starts by presenting a short background on the techniques of Internet measurement and lists some of the prominent (active) measurement plat-

forms including SamKnows, MONROE, SpeedCheker, and CAIDA Ark. Then it discusses the evolution of the web, the factors affecting the web performance, web QoE models, and the available tools for web performance measurement. The web has evolved from a simple content transmission system to a complex ecosystem with rich media-contents. The changes in the web ecosystem are not limited to the content only but also in the content provisioning, and the protocols in the web stacks. The changes in the ecosystem have an impact on the web performance. The web performance is relevant not only for the end-user but also for the content and service providers. The web performance is affected by the transport protocol, the name resolution and content encryption, client device computational power, the web page complexity, the browser rendering engine, etc. In order to improve the web performance several techniques has been proposed including content caching, improvements in the protocols, content prioritization, and content compression.

The web QoE is the overall satisfaction of the end user while browsing a web page, as perceived subjectively. Factors that emanate from the system, the user, the content, and the context of use affect the user-perceived browsing experience. The key parameters for measuring web QoE are the waiting time to get the web content and the usability of the web page. Several metrics are available to objectively measure the web QoE. The metrics are either time-instant metrics or time-integral metrics. Time-instant metrics are measured by looking at the occurrence of a specific of a particular event during the browsing session, for instance, the arrival time of the first byte, the time of the first paint in the screen, the time of the first meaningful paint, ATF time, or the completion time of the web page download. On the other hand, time-integral metrics are computed by integrating the timing of several events that happen in the web page browsing process. The Speed Index, Byte/Object Index, and Ready Index are metrics of this class. These objective metrics can be mapped to the perceived QoE that is expressed in terms of MOS. The models for mapping the objective metrics to the perceived QoE are either expert models (logarithmic and exponential modes) or data-driven models using machine learning techniques. In the next chapter, we present the metrics and methods we proposed and the tools we implemented to measure the web performance and QoE.

3. Metrics, Methods and Tools

All actors in the web ecosystem from the ISPs to the content providers would like to understand if the users enjoy browsing the web. This is because understanding the performance of accessing the web helps to deliver a better service quality and maintain the user base [140]. Delivering a service with a better quality of service is directly linked to reducing customer churn and increasing the business revenue [135].

To understand the user-perceived QoE and to improve the service quality, service and content providers need to measure and monitor the quality on their customer premises. For instance, ISPs collect network traces at their backbone network and analyze the traces to identify whether a customer's poor browsing experience is because of a network problem, a problem on the content provider side, or an issue related to the client device. However, the traces collected at the backbone network do not reveal the real end user experience. This is due to the complex nature of the web ecosystem, and the traces do not have information about the end-user device. The complex nature of the web ecosystem makes measuring the browsing performance and QoE challenging. This is due to the diversity of webpages, heterogeneous types of devices and browsers, choice of metrics (*e.g.,* network-centric, browser-centric, and user-centric), and lack of well-established metrics [180].

Moreover, the quality measurements on the service and content providers' side may not be important for the end users. The end users care more about the QoE of the system that they perceived subjectively (see Section 2.4). Understanding the end user QoE is paramount for the service providers to deliver a quality service [139].

A number of studies have proposed various tools to measure the web browsing QoE. Most of the proposed tools to approximate the web QoE measure various metrics that are collected at the network or application

layer. As mentioned in Section 2.4.3, new metrics such as the ATF time and Speed Index have been proposed to better approximate the web QoE.

The contents within the *above-the-fold* area or visible part of the webpage are the important part of the webpage for the user to judge whether or not a webpage has been downloaded and rendered [104]. The rest of the contents in the webpage may still continue to download after the contents in the above-the-fold area are displayed. However, the contents out of the above-the-fold area do not change the appearance of the webpage. Therefore, to approximate the perceived QoE, measuring the time the users see the content in the above-the-fold area is more appropriate than measuring the occurrence of browser events. Browser event-related metrics such as the PLT may under- and over-estimate the QoE. For instance, content in the above-the-fold area loaded by JavaScript or Flash occurs after the `onLoad` event. In this case, the PLT underestimates the web QoE.

This chapter focuses on the ATF time and methods to approximate it. We first present Webget – a web latency measurement tool (3.1). Then we describe our methodology to approximate the ATF time (3.2). Then, we discuss the implementations of the tools that we developed to measure the ATF time on a different platform and scale (3.3). Along with describing the design and implementation, we present the validation of our methodology to measure the QoE time. The content of this Chapter is taken from Publications I, II, III, and IV.

3.1 Webget: Measuring Web Latency

Webget is a software that records the DNS lookup time, TTFB, the download time, the number and the size of each static object that makes up the website. Webget runs a maximum of eight concurrent connections, and up to eight parallel threads per domain. We chose this setting to match the behavior of the user-agents that we have used in the past. We have not updated the number of the parallel threads so far because it would cause a significant change in the Webget result. The goal is to keep the parameters consistent across the longitudinal (several years) data collection to avoid fragmenting the data into smaller samples by not tweaking the parameter space over time. Webget does not execute scripts. It does not download nor take into account the dynamic objects, which are common in modern websites.

3.2 Above The Fold (ATF) Time Approximation

The ATF time is the time until the content in the above-the-fold area of the webpage stops changing and reaches a final stable state. The ATF time can be approximated using different approaches. One way to approximate the ATF time is by monitoring the pixel changes in the visible part of the webpage and detecting when it stabilizes [136]. That is, by estimating the time of the last pixel change in the above-the-fold area. We refer to this method of approximating the ATF time as a pixel-wise comparison approach. Another method to approximate the ATF time is by using the resource timing information that the browsers provide [11]. We refer to this approximation method as a browser heuristic-based approach.

3.2.1 Pixel-wise comparison approach

To approximate the ATF time of a website using the pixel changes in the current viewport of the browser, the webpage download progress is recorded as a video sequence for a given amount of time. The recorded video is converted into a series of screenshots (bitmap images) at a given frequency (we found that a frequency of 100 ms is reasonable [136]). Then, the pixel changes between the consecutive bitmap images are calculated. When there is no pixel change between consecutive X number of images for a certain stabilizing threshold (*i.e.,* X/10 seconds threshold), the webpage is declared to be loaded and rendered. Our study showed that in mobile networks, webpages stabilize within a three seconds threshold [181]. Hence, the ATF time is the time from the beginning of the browsing session until the time at which the last pixel change event is observed. In the remainder of this thesis, we refer to the ATF time approximated using this method as ATF_p.

Algorithm 1 shows the pseudocode of the pixel-wise comparison approach of ATF time approximation. The algorithm takes the list of screenshots, the pixels in the screen, the frame rate at which screenshots are captured, and the web page stabilizing threshold as inputs. The algorithm first computes the pixel differences between for all two consecutive screenshots and stores the differences in a dictionary (line #6 to #9). Then, it iterates through all the keys in the dictionary that contains pixel differences (line #10), and calculates the percentage of the pixels that have changed from the screen (line #11). It considers as a rendering and painting event has happened in the screen when the percentage of the pixel changes on the

```
input  : LIST S ← list of screenshots
         P ← Pixels in the screen
         r ← frame rate
         t ← stabilizing threshold
output : Approximated ATF time
DICT pixelChanges ← {}
lastChange ← 0
begin ← -1
end ← -l
atf ← 0
for i ← 2 to length(S) do
    diff ← comparePixelDifference( S[i], S[i-1])
    pixelChanges[i] ← diff
end
for j in pixelChanges .keys() do
    delta ← 100 × pixelChanges[j] / P
    if begin < 0 then
        begin ← j
        lastChange ← 0
    end
    if delta ≥ 0 then
        end ← j
        atf ← ((end − begin ) / r) ×
        lastChange ← 0
    end
    if lastChange ! = -1 and delta == 0 then
        lastChange ← lastChange + 1
    end
    if lastChange ≥ t × r then
        break
    end
end
```

Algorithm 1: ATF time approximation using pixel-wise comparison approach.

screen is greater than 0 (line #18). When a rendering and painting event has occured on the screen, the algorithm computes the ATF time by taking the number of screenshots from the last a rendering and painting event observed until the current one dived by the frame rate r (line #18). From a given point where rendering and painting event is observed, the algorithm iterates until the number of consecutive screenshots is less than the web page stabilizing threshold times the frame rate (line #24 to #26). It finally returns the approximated ATF$_p$ time.

3.2.2 Browser heuristic-based approach

The W3C consortium has standardized APIs for web browsers which helps retrieve resource, navigation and performance timing information of websites. The resource timing API provides detailed network timing data and the location of each object within a webpage. The ATF time can be approximated by leveraging the information from the browser, without requiring image processing. The approach works as follows. It takes the maximum of the download time for the HTML documents, JavaScript and style sheet files, and the images located at least partially within the above-the-fold area of the browser [181, 11]. The ATF time approximation can be formally defined as follows:

$$AATF = \max_{o}\{T_o | o \in \mathcal{H} \cup \mathcal{J} \cup \mathcal{C} \cup \mathcal{I}_{ATF}\} \tag{3.1}$$

where T_o is the loading time of object o, and $\mathcal{I}$ is be the set of all images, $\mathcal{I}_{ATF}$ is the subset of images whose coordinates are at least partially above-the-fold, $\mathcal{H}$ is the set of all HTML documents, $\mathcal{J}$ is the set of all JavaScript HTTP requests and $\mathcal{C}$ is the set of all CSS requests.

The first task in this approach is to identify the image locations. The images that are added to the webpage using the `<img>` tag can be easily identified. However, some websites add images as a background to the webpage and manipulate them using style sheets. For such cases, the style sheets need to be evaluated to identify the image locations as adjusted by the style sheet. In the rest of this thesis, we refer to the ATF time approximated using this method as ATF$_b$.

3.2.3 Challenges and limitations

Both of the aforementioned approximations have their limitations. As such, one approach could better approximate the ATF time for certain kinds of

websites, while the other approach may underestimate or overestimate it or vice versa. The main challenges are the following:

In the pixel-wise comparison approach, the webpage may not stabilize for different reasons. For example, the page may contain animated contents or an auto-play enabled video that changes frequently. As such, it might be difficult to detect when the visible part of the webpage has completed loading. This makes it challenging to approximate the ATF time using the pixel-wise approach. Finding an optimal cutoff point for the website stability is also challenging. That is, the website stabilizing threshold may depend on the network technology and the end-user device. Another limitation of this approach is that it is computationally expensive (the ATF time computation takes 45 seconds of CPU time and 290 MB of memory [136]) and its applicability is mainly limited to a laboratory settings.

In the browser heuristic-based approach, sometimes, it is difficult to identify the exact location of some types of objects (*e.g.,* objects created by a script execution) on the webpage. Web designers also use different ways of adding images to a webpage, *e.g.,* using the `<img>` tag, adding the image as a background using style sheets, or using custom tags specific to the website. For instance, in the experiment we conducted in 2019, we observed that Facebook uses the `<img>` tag for placing images that are part of the static webpage layout and uses CSS and AJAX to load dynamic images such as user-uploaded photos. These images are not detectable using the resource timing API or by evaluating the style sheet. As a result, it is challenging to identify images that are added to the webpage using a custom tag. Moreover, the resource timing API may not always be accurate enough to provide the download time for resources [180]. For security reasons, unless it is explicitly allowed by the server to provide, the browser timing API does not provide all the timing information for cross-origin objects [182]. Moreover, the implementation of the timing events is browser-dependent. Because of this, the ATF time of websites approximated using browser timing API may vary between browsers. In Chapter 4, we use both metrics and evaluate the ATF time of different websites.

3.3 Implementation

We implemented both aforementioned methods of ATF time approximation as different tools that can be applicable to lab settings and real-world mea-

surements. First, we designed *WePR* (which stands for *Web Performance and Rendering*), an automated web performance measurement system, and implemented the pixel-wise comparison approach as part of the system [183, 136]. Then, we implemented a browser heuristic-based method as a browser extension (the current implementation works for Chromium-based browsers including Google Chrome and Brave) [11]. Lastly, we combined both methods in a command line tool [181]. In this section, we describe the measurement system and the implementations of the tools.

3.3.1 *WePR*: Web Performance and Rendering

Although various tools are available (see Section 2.5) to quantify the web browsing performance, most of them either do not consider a wide-range of metrics at different layers in the communication stack or are not suitable for conducting measurements on a large scale. Therefore, one of the goals of this thesis is to design and develop a scalable measurement system that allows us to measure the web performance and QoE using different metrics including short web flow latency metrics, webpage complexity metrics, PLT and the ATF time. To this end, we designed and implemented *WePR* , a web latency and rendering measurement system composed of a number of components.

System design and architecture. *WePR* has a distributed architecture where some of the components are deployed in a central location (*e.g.,* in a data center), while the probes are located at the Internet gateways of different network users such as in home networks, enterprise networks or corporate networks. The management of the probes is usually handled by measurement companies such as the SamKnows measurement platform (see Chapter 2.1.4).

We designed a distributed architecture comprising decentralized (lightweight) measurement probes and centralized processing and storage components. First, the measurement probes (in our case the SamKnows probes) that are deployed at the Internet gateways to collect the web performance measurements from customer premises are often lightweight devices. These devices have resource constraints and do not have the capability of running resource-intensive applications such as browser rendering engines. As a result, it is not possible to run a measurement tool and collect application level performance metrics from these devices. Hence, we need to offload the resource intensive computations such as approximating the ATF time to a parsing server and rendering server. Second, assuming the measure-

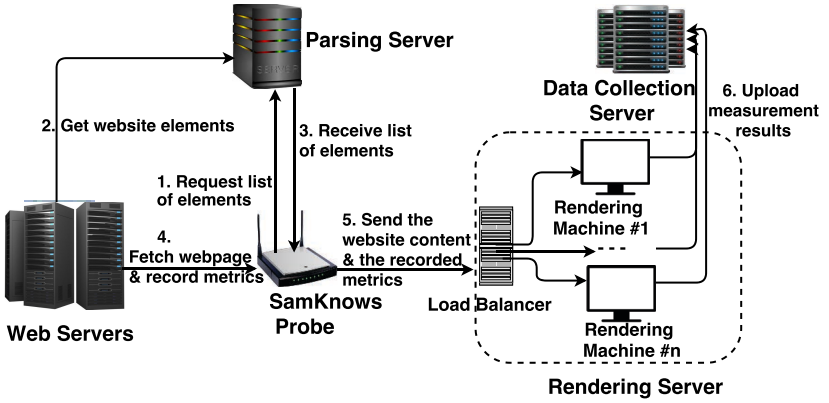

Figure 3.1. The distributed architecture of *WePR*. The SamKnows Probes are located in the customers premises, the parsing and rendering servers are located in data centers.

ment probes are powerful devices with sufficient resources to perform the computations (*e.g.,* using a regular volunteer PC), it is not always possible to alter the user settings and install the extra necessary software. For instance, in order to estimate the ATF time using the pixel-wise approach, we need to install a screen recorder (*e.g.,* FFmpeg) and image processing (*e.g.,* ImageMagick) tools. Therefore, offloading different functions of the system into different components is essential to estimate the ATF time using a pixel-wise comparison approach.

Figure 3.1 depicts the distributed architecture of WePR. The major components of the system are the parsing server, the SamKnows probe, load balancer, rendering server and data collection server. The SamKnows probes are located at Internet gateways geographically distributed across the globe. The parsing server, the load balancer, the rendering server, and the data collection server are located within a data center. Each of the components is described in detail as follows.

SamKnows probes. These are OpenWRT-based embedded measurement boxes which run various performance and reachability measurement tests (see Section 2.1.4). We leverage the SamKnows probes to run a web performance measurement test – *WebPerf* – which measures the browsing performance from multiple vantage points.

WebPerf is the software that measures web latency metrics. It downloads the actual web objects and along with the other measured metrics, it pushes them to the rendering server. WebPerf downloads all the objects (including those generated by JavaScript execution) based on the list it receives from the parsing server. The metrics it measures include the DNS lookup time, the number of messages exchanged during the name

resolution, the time to establish TCP connection, the time to perform a TLS handshake, the HTTP header size, the number of HTTP redirects, the time elapsed due to the HTTP redirect, the time to receive the first byte of the payload, the download time for each object, and the number and size of the objects.

The WebPerf test takes the URL of the webpage and sends a request to the parsing server to get the list of URLs of the objects that make up the website. Once it receives the list of URLs, it fetches the objects. During the download process it measure the aforementioned metrics. For handling the HTTP downloads and extracting the HTTP header information, we have developed a library. The library is designed to intercept the request processing at the key points in the HTTP download process. WebPerf uses the hooks of the library to record the timestamps, extract header information, and save the received data. *WebPerf* is written in C and can be cross-compiled for any Unix-based platform. The software is open-source and available at [184].

SamKnows also has its own web performance measurement tool called **Webget**. Webget measures web latency metrics for static objects of the webpage. The metrics include the DNS lookup time, TTFB, download time, the number and the size of static objects on the website. While Webget considers only the static objects on the website, WebPerf considers both static and dynamic objects of the website including those created as a result of script execution.

Parsing server. This entity parses the homepage of a website and retrieves the URIs of each object (both static and dynamic) that make up the website. Once it receives the parsing request (the URL of the website) from the WebPerf test, in addition to parsing the DOM tree of the homepage, it executes JavaScript using *PhantomJS*. After parsing the DOM tree, it records the target URIs of the objects that need to be fetched to build the webpage. When this process is completed and all the Universal Resource Indicators (URIs) of the objects comprized the web page are determined, it sends them to the SamKnows probes. Each probes fetches the objects based on the list and measure the performance of the website from multiple vantage points. The parsing server can be deployed in the probes themselves if the probes are not resource constrained and are capable of running a (virtual) browser rendering engine.

Rendering server. This is the component that recreates the website based on the objects and measurement results it has received from the

probes and calculates the ATF time using the pixel-wise comparison method. It comprises a load balancer and one or more rendering machines. Since the ATF time computation involves image processing, it requires a substantial amount of computing resources. Thus, in order to serve multiple rendering requests simultaneously from a large number of probes, we need to use multiple rendering machines. The load balancer is responsible for managing the incoming rendering requests from the probes and for distributing to the available rendering machines with a round-robin mechanism. We implemented the load balancer using *HAProxy* [185], which provides a load balancing and proxying service for TCP- and HTTP-based applications.

To calculate the ATF time, the rendering machine executes two different applications. These are the playback service and rendering manager.

- The **Playback service** emulates a DNS server and an HTTP server. It responds to domain name resolution and HTTP requests. The playback service responds based on the metrics recorded by *WebPerf* at the probes and pushed along with the web objects. It adjusts the response time and transmission rate to match the network delay observed at the probes where the performance measurement is conducted. Throttling the response and the transmission rate to the *WebPerf* measurement result helps ensure that the rendering server follows the same network QoS performance as observed by the probes. In our implementation, the playback service resolves a given name to a local host. The playback service retrieves the inputs of the response from the local storage (*i.e.,* the objects are fetched and pushed by the probes and stored in the rendering machines). The playback service is written in C.
- The **Rendering manager** is the application that calculates the ATF time. It opens a (virtual) browser and issues a request to a given URL. The DNS resolution and HTTP responses are handled by the playback service as described above. Once the browser gets the responses from the playback service, it renders the requested webpage. The rendering manager records a video (10 frames per second) of the browsing session and approximates the ATF time using the procedure described in Algorithm 1. We use Selenium and ImageMagick to automate the screen recording and image processing, respectively.

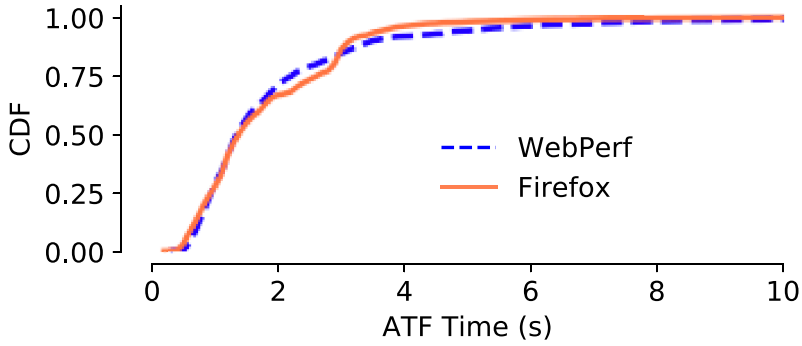

Figure 3.2. The ATF time of ten websites when the contents are fetched by the Firefox browser and by *WebPerf*. The websites show similar rendering behavior regardless of the tool (*WebPerf* or Firefox) used to fetch the contents. Note: the experiment was performed 150 times for each web page.

3.3.2 Validation of the WebPR system

To validate the pixel-wise comparison approach of the ATF time approximation, we measured the rendering behaviour of ten non-HTTPS websites in two cases. That is, when the content was fetched by *WebPerf* and rendered using Firefox, and the other case when the web content was downloaded and rendered using Mozilla Firefox. We performed the experiment from the same laptop connected with a university WiFi network. The specifications of the laptop were: 8GB RAM, quad-core Intel processor (2.3GHz each), and running the Ubuntu 16.0.2 LTS operating system. We set a similar configuration (*e.g.,* the maximum number of parallel connections, number of concurrent threads per server, and User-Agent) for *WebPerf* and Firefox. On each consecutive run, we also cleared the cache before fetching the web content.

Figure 3.2 shows the distribution of the ATF time of the websites when the content was fetched by *WebPerf* and Firefox. The experiment was run 1500 times. As can be seen the websites have similar ATF time when the content was fetched by *WebPerf* and the Firefox browser. In 50% of the cases, the difference between the ATF time of the websites when the two different tools are used to fetch the content is minuscule. This implies that the websites have similar rendering behaviour irrespective of the tool (*WebPerf* or Firefox) used to fetch the contents. In 25% of the cases, the ATF time was shorter (by 400 ms) when Firefox is used to fetch the content. In the 25% of the cases, the ATF time was shorter (by 500 ms) when *WebPerf* was used to fetch the webpage. One of the reasons for this difference could be the load on the web server. The web servers could

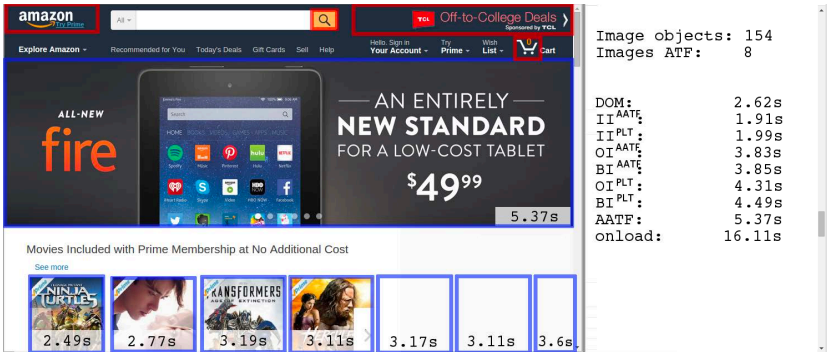

Figure 3.3. Illustrative example: *Time-instant* metrics show that whereas DOM loads at 2.62s, all objects above the fold are rendered on or before approximated ATF=5.37s and then the page finishes loading at PLT=16.11sec. By definition, *Time-integral* metrics are even shorter BI^{AATF}<BI^{PLT}< approximated ATF, hinting that ATF may be significantly off with respect to timescales relevant to the user perception.

vary at difference times, which could result in a delay variation in the response. Another reason could be the latency difference due to variation in the network path while fetching the objects using *WebPerf* and Firefox. Due to network routing changes at different times, *WebPerf* and Firefox may traverse different network paths which could result in a variation in the latency.

3.3.3 Approximate ATF Chrome extension

We implemented the browser heuristic-based approximation method as an open source browser extension for Chromium-based browsers such as Google Chrome and Brave [186]. The extension script executes after the browser triggers the `onLoad` event. We used jQuery to detect the visible DOM objects. For each of the objects, we detected the dimension and the location in the webpage. Using this information and the dimension of the browser window (which we obtained using JavaScript), we determined the images that were within the above-the-fold area. We used the `Window.performance` API to retrieve the name, type, and timing information about each object within the webpage. We compared the `src` field of the DOM object to the `url` field of the HTTP request to match HTML objects to their corresponding timing information. Then we calculated the ATF time using Equation 3.1. In addition to approximating the ATF time, the extension also calculates other metrics such as Object and Byte Index with different cut off points. The extension has different configuration options. For instance, it has the feature to set the deadline for executing the scripts in the case the browser `onLoad` event is not triggered.

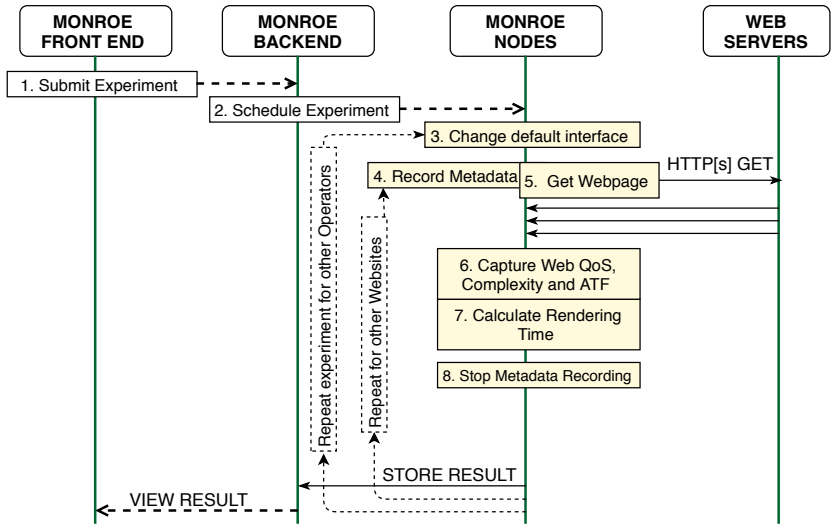

Figure 3.4. Sequence diagram of the experiment using WebLAR tool in MONROE measurement platform.

Figure 3.3 shows an example of the execution of the extension used in the Google Chrome browser when browsing an Amazon webpage. As can be seen from the Figure, the Webpage has a total of 154 images, of which 8 are located within the above-the-fold area (circled in blue). In this illustration example, the ATF time of the webpage is basically the download time of one of the images within the above-the-fold area. The approximated time is shorter than the PLT of the webpage.

3.3.4 WebLAR: A tool for measuring Web Latency and Rendering

WebLAR is a tool that we designed, implemented, and open-sourced to measure the web QoE using both the pixel-wise comparison and browser heuristic-based approaches. In addition to approximating the ATF time, it measures other time-instant QoE metrics including PLT and TTFB, and web complexity metrics such as the number and the size of the objects within the webpage. Whenever available, it also records metadata information about the network and the device. Unlike WePR, WebLAR performs the image processing to calculate the ATF_p time directly on the end-user device without requiring to send data to another machine. We implemented the tool primarily for measuring the web QoE in cellular networks over the MONROE measurement platform. However, it can be deployed to other measurement platforms and used with minimal configuration changes.

Workflow. Figure 3.4 shows the sequence of operations for web QoE mea-

surement using the WebLAR tool on the MONROE measurement platform. The MONROE measurement platform provides a web and command line interface to submit web measurement tests. The user submits an experiment in the MONROE front end (#1 in Figure 3.4). When a user submits an experiment, the user needs to set the parameters of the test in the user interface. For instance, the user needs to provide a link to the repository of the Docker container for the experiment, the number of probes that the experiment is supposed to be performed on, the number of mobile operators, etc. Then, at the second stage (#2), the MONROE back-end schedules the test at the selected probes specified by the user. It also triggers the experiment execution at the probes based on the parameters that the user provided in the user interface. Upon the start of the experiment, a probe checks whether or not the Docker container is available locally in the probe. If not, it fetches the Docker image from the remote docker repository.

When the experiment starts, first the default network gateway of the probes is changed to one of the available cellular or WiFi network interfaces (#3). Then, WebLAR starts recording metadata information about the network and the probe (#4). Next, it immediately opens Google Chrome (version 62) using Chromedriver and starts fetching the website (#5). The browser starts in an Incoginto mode to ensure that no content from browser's caches is fetched. While fetching the webpage, WebLAR records a video of the webpage download session. Once the webpage download is complete, it uses the Chrome API to capture the timing, performance, and navigation information about the page (#6).

Once the browsing session is complete, the recorded video is converted into a series of bitmap images at every 100 ms (10 frames per second). Using these images, WebLAR calculates the ATF_p time (#7). The captured timing and performance information is also used to calculate the ATF_b time, PLT, TTFB, DNS lookup time, TCP connect time, TLS handshake time, etc. Then, it stops recording the metadata information (#8). Finally, it annotates the web QoE measurement result with the recorded metadata information and sends to the MONROE back end. When the probe has more than one cellular interface, WebLAR repeats the steps from #3 to #8.

Implementation. Since our objective is to develop a tool that runs without a graphical user interface and measures the user browsing QoE, we developed WebLAR as a command-line tool. WebLAR is implemented as a set of modules for the following functions:

The first module is responsible for setting up the network gateway inter-

face, starting or stopping the metadata recording, aggregating the results, and sending them to the data storage drive. It also sets up a virtual screen using the X virtual framebuffer (Xvfb) [187] that enables us to run the browser without requiring an actual display device. This module is implemented using shell and python scripts.

The second module is the one that opens the Google Chrome browser using the Chrome driver in the virtual screen configured by the first module and browses the webpage. This module also records a video of the browsing session and captures the timing and performance information about the webpage using the Chrome API. Once the browsing session is complete, it calculates the ATF_b time, PLT, TTFB, etc. We implemented this module using Java. We used Selenium to automate the video recording in a virtual screen.

The third module uses `FFmpeg` to convert the recorded video to a series of bitmap images. It uses `ImageMagick` to compare the pixel difference between the consecutive screenshots. We used shell scripts to automate this image processing. Then, it calculates the ATF_p time by using Algorithm 1. We implemented Algorithm 1 in Python. Since MONROE uses `Docker` as a virtualization technology, we packed the modules and the necessary libraries as a single `Docker` container.

The validation we performed for the methods, WePR system and the Approximate ATF tool ensures that the approach we used in WebLAR is valid. Thus, we do not need extra validation for WebLAR.

3.4 Summary

This chapter provides an overview of the metrics and methods we proposed along with the implemented tools to measure the ATF time of web pages. The ATF time is the time until the content in the above-the-fold area of the web page is downloaded and rendered. The ATF can be approximated using two methods: the pixel-wise comparison approach and browser heuristic-based approach. In the pixel-wise comparison method, the ATF time is computed by examining the pixel changes within the current viewport of the browser while a web page download is progressing. Instead, in the browser heuristic-based method, a browser timing API is used to retrieve the timing information of the resources within the above-the-fold area. The ATF time is computed by taking the maximum download time of the images within the above-the-fold area, HTML files, style sheet files, and

JavaScript files.

These methods are implemented using piece of command line software and a browser extension. *WePR* is a measurement system that implements the pixel-wise comparison approach of ATF time computation from the command line. We also implemented the browser heuristic-based approach as a browser extension for Chromium-based browsers including Google Chrome and Brave. Moreover, we implemented *WebLAR*, which is a command line tool to measure the ATF time using both the pixel-wise comparison and the browser heuristic-based approaches. In the remainder of this thesis, we present the analysis of the measurement data collected using the developed tools. In the next Chapter, we provide the result of the web latency and rendering performance in fixed-line networks.

4. Web Latency and Quality of Experience (QoE) in Fixed-line Networks

This chapter focuses on web latency and QoE, and discusses the main findings from the analysis of a longitudinal dataset. The measurement dataset wass collected using the *SamKnows* measurement infrastructure [29]. We first describe the dataset in Section 4.1. Then, in Sections 4.2 and 4.3, we present the overall findings on web latency and the impact of cache presence on web latency, respectively. Following that, in Section 4.4, we discuss the longitudinal observations on web latency. In Section 4.5 and 4.6, we present the impact of throughput on web latency, and the web latency between different regions, respectively. Finally, Section 4.7 discusses the findings on the web QoE. The content of this chapter is based on Publications [I] and [IV].

4.1 Dataset

We deployed the *Webget* test on 182 SamKnows probes distributed globally. The probes were located in more than 70 origin ASes, covering 28 countries. The test measures the performance of specific webpages of three popular websites. The webpages are:

- `www.facebook.com/policies`
- `www.google.com/mobile`
- `www.youtube.com`

In the rest of this chapter, we refer to these webpages as Facebook, Google and YouTube, respectively. These webpages are measured at every hour. We run the measurement for 3.5 years (January 2014 - July 2017).

We considered the popularity, content consistency and size of the webpages as criteria for our choice. We selected webpages that showed meaningful content without requiring user interaction. Since our objective in

this measurement was to study the longitudinal aspect of web latency, we have chosen webpages that have relatively consistent content across the measurement period. Moreover, as we ran the measurement from volunteers' homes and repeated them every hour, we took the size of the webpages into consideration not to overwhelm the users' home traffic with our measurement traffic. Note that our aim in this measurement was not to compare the performance of webpages. Rather our objective was to understand what factors contribute to the web performance across ISPs and regions. In the subsequent sections, we presented the key observations from the analysis of this dataset.

4.2 Web Latency

Before diving into the performance analysis, we describe the complexity (in terms of the number and the size the objects) of these webpages and the number of the servers used to host the contents of the webpages. Google had relatively the same complexity for the three years, and it reduced the number and size of objects since 2017. When we consider YouTube, in 2015, it refactored the homepage. As a result, the number of static objects on YouTube's homepage has decreased. In July 2017, Facebook introduced privacy basics and updated its terms and services. Due to this, the number of static objects on Facebook's policy pages has increased. Until then Facebook's policy had only a single static object. Throughout the duration of the measurement period, Facebook had fewer and smaller objects than the other two. In 80% of the cases, Google has the highest number of objects. Moreover, these webpages had variations in terms of the number of servers that hosted the contents. For instance, in the median case, Facebook used a single server to host the policy page. While in the median case, Google used two (from 2014 to 2016) and four (in 2017) servers, and YouTube used three servers (from 2014 to 2015), two servers (in 2016) and a single server (in 2017) to host the contents of the webpages. Note that, we identified the number of servers by counting the number of DNS lookups performed. However, the content might be hosted on virtualised servers in data centers that share the same public IP endpoint, *e.g.,* via reverse proxying.

The latency metrics we considered in this measurement include DNS lookup time, TTFB – the time to receive the first byte of the payload – and download time – the time from the beginning of the request until

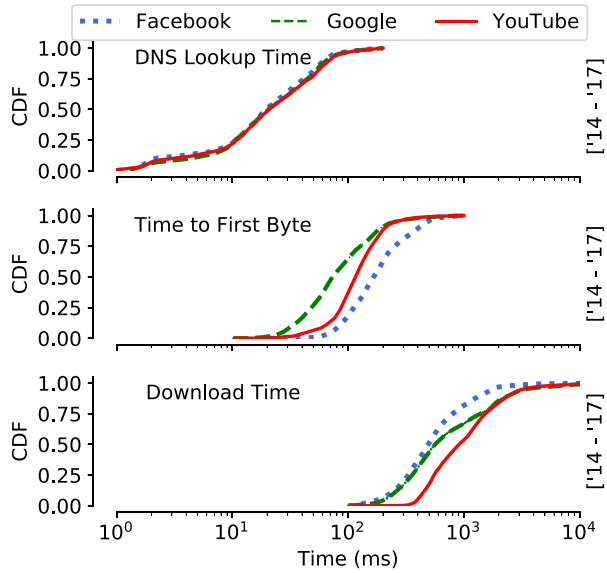

Figure 4.1. CDF of the performance of the three popular websites in terms of different metrics.

receiving the last byte of the webpage. We measured the time required to resolve the domain name of each object on the webpages and the time to receive the first byte of the payload. Since the webpages contain multiple objects, we took the average of the DNS lookup and the average TTFB of the objects that make up the webpage as the webpage's DNS lookup time and TTFB, respectively. The download time is the time until the client receives all the objects. Note that, in all of the discussions in this section, the download time does not include the browser rendering engine's processing and rendering time.

Figure 4.1 shows the distribution of the average DNS lookup time, the average TTFB and download time of the webpages. As can be seen, in 55% of the cases, the three webpages had similar performances in terms of the DNS lookup time. However, in the rest of the cases, Facebook had a shorter (10%) DNS lookup time than Google and YouTube. The reason for this variation could be attributed to the number of servers that the clients need to contact. A client needs to contact a maximum of two servers to fetch Facebook's policy page. Whereas to fetch Google's and YouTube's webpage the client has to contact two to four and one to three servers, respectively.

We also observed a difference in the average TTFB of the webpages. In the median case, Google had a shorter average TTFB than the other two webpages. The difference in the average TTFB between Google and YouTube (which are supposed to share the same CDN infrastructure) was

significant. That is, in 50% of the measurements, Google had 55% improvement compared to YouTube. A reason for this performance variation in the average TTFB is the difference in the IP path length between the clients (SamKnows probes) and the web servers (content replica).

Although making a performance comparison between the webpages was not our goal, we investigated the difference in the download time of the webpages. The objective was to understand how the complexity of the webpages contributed to the download performance. As can be seen in Figure 4.1, Facebook loads faster than the other two webpages. However, in 90% of the cases Google loads faster (62% at the median) than YouTube. As mentioned before, Facebook has fewer and smaller objects, and indeed it has better download performance. However, a previous study [99] shows that the complexity metrics are not the only determining factors for download performance. Our measurement confirms that the number and the size of objects do not directly affect the webpage download performance. For instance, despite the fact that Google has the higher number and larger size of objects than YouTube, it has a shorter download time than YouTube.

4.3 Web Latency and Cache Presence

Before we discuss the impact of cache availability on the web latency, let us first see how these popular websites distribute their content. As we discussed in Section 4.2, the number of servers that host the contents of the webpages varies depending on the webpage. In addition to knowing the number of the servers, we sought to understand the location of the content replicas. In doing so, we performed `traceroute` measurements to study the number of IP path lengths from 100 SamKnows probes (a subset of the 182 probes) towards Facebook, Google and YouTube services. Figure 4.2 shows that from the location of the probes, Facebook (F) – shown in black dotted line – has a longer IP path lengths than Google (G) and YouTube (Y).

We used the `traceroute` measurement to identify whether or not the webpage had caches at ISPs or other ASes. That is, when the probe's AS number is the same as the destination AS number, it is identified as a cache at the ISP. When the destination AS number was different from the probe's, Google's or Facebook's, we concluded that the webpage had a cache at another AS. Applying this heuristic, we did not find any Facebook caches at the ISPs. In 41% of the cases we found Google caches at ISPs, this was true for *i.e.,* 25% of the cases from the ISPs that the probes were

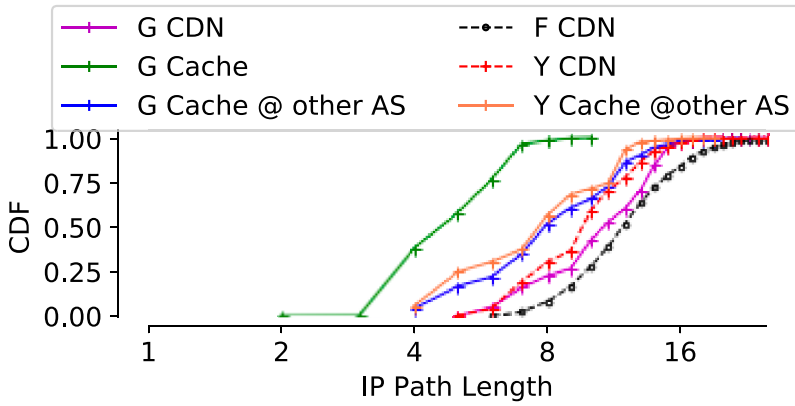

Figure 4.2. The IP path length from 100 SamKnows probes towards Facebook (F), Google (G) and YouTube (Y). In 41% of the cases, the probes get Google caches from the ISP.

connected to and 16% from other ASs. Moreover, in 24% and 10% of the times, we found a YouTube cache at ISPs and other ASs, respectively.

As discussed in Section 4.2, although the domain name resolution time of Facebook is similar to that of Google and YouTube, the average TTFB of Facebook is longer compared to the other two webpages. The reasons for this could be the absence of Facebook caches at ISPs and the longer IP path length towards the Facebook CDN. As mentioned before, we did not find the Facebook caches at the ISPs or other ASs. A previous study [188] also shows that Facebook has a longer AS path length from RIPE Atlas probes [28] compared to other popular CDNs such as Google. For instance, 60% of Facebook's content is reachable in 2 AS hops, while more than 55% and 80% of Google's content can be reached with a maximum of 1 and 2 AS hops, respectively.

4.4 Longitudinal View of Web Latency

In this section, we look the evolution of web latency over the course of our measurement period. Figure 4.3 shows the monthly median of the DNS lookup time of the webpages over the three-year period. As can been seen, the DNS lookup time of all webpages improved over time. For instance, from 2014 to 2016, in 50% of the measurements the DNS lookup time for Facebook, Google, and YouTube improved by 59%, and 56%, 61%, respectively.

Figure 4.4 shows the monthly median of the download time of the webpages over time. It can be seen that in the median case, the download time

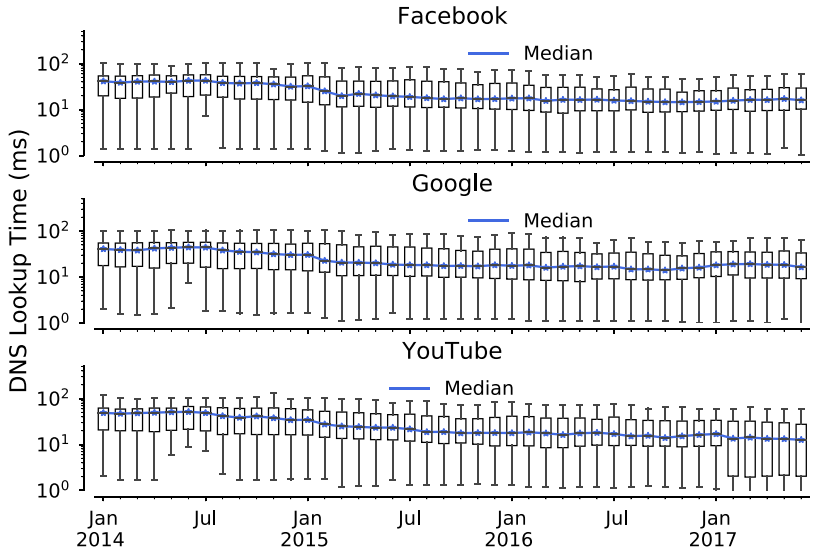

Figure 4.3. The monthly median of the DNS lookup time of webpages over time. The DNS lookup time of webpages has improved over time.

for Google shows an improvement during the course of our measurement. However, the other two webpages do not show a clear trend over time. Due to the increase in the number of static objects on the webpage, Facebook's download time has increased since July 2014. Since the middle of 2014 the download time for Facebook has not shown a significant change. Moreover, until middle 2015 YouTube had both increasing and decreasing trends in the download time. However, around July 2015, YouTube refactored their in the homepage which reduced the number of objects by a factor of four. As a result the total size of the static objects on YouTube's landing page has substantially decreased. This could be one of the reasons for the improvement in YouTube's download time from the middle of 2015. Note that, the Time to First Paint (TTFP) of the webpages shows a similar behavior trend as the download time.

4.5 Web Latency and Broadband Speed

A previous study shows that the broadband speed of the client affects the web browsing performance [189]. To understand the evolution of (if any) the throughput that the probes are getting, and to verify whether there is a correlation between web latency and the throughput, we performed a throughput measurement on the SamKnows probes. The web latency test and the throughput test ran independently, but we took hourly averages of

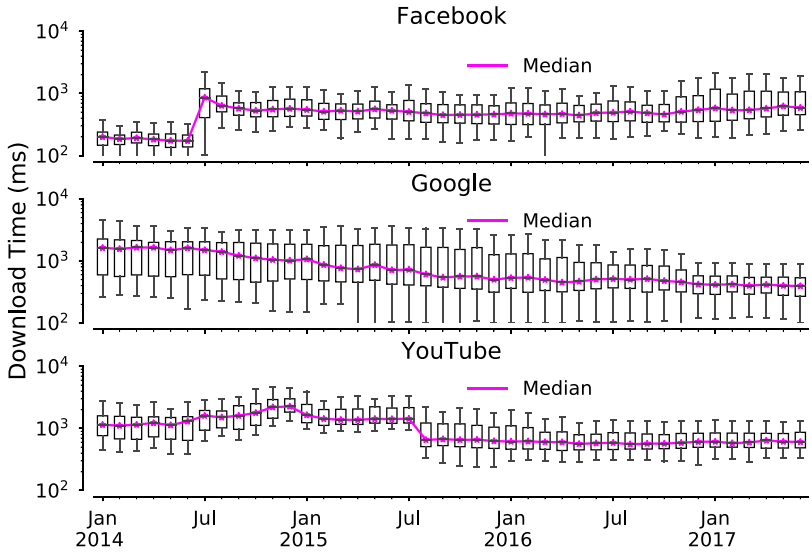

Figure 4.4. The monthly median of the download time of the webpages over time.

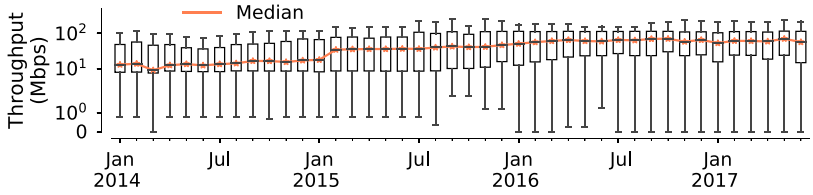

Figure 4.5. The monthly median of the achieved throughput as witnessed by the probes over time.

the results from both tests for each probe.

Figure 4.5 shows the time series of the monthly median of the achieved throughput as seen by the probes. As can be seen, there has been an improvement in the broadband speed during the course of the measurement period. As we have discussed in the previous section, the download time for Google improved over time, whereas the other two webpages did not exhibit an improvement in the web latency, even though there was an improvement in the achieved throughput. This shows that improvements in broadband speed does not necessarily yield a lower web latency and better browsing experience [167]. That is, once capacity is no longer botleneck (or above a certain threshold), the web latency is not affected by the bandwidth.

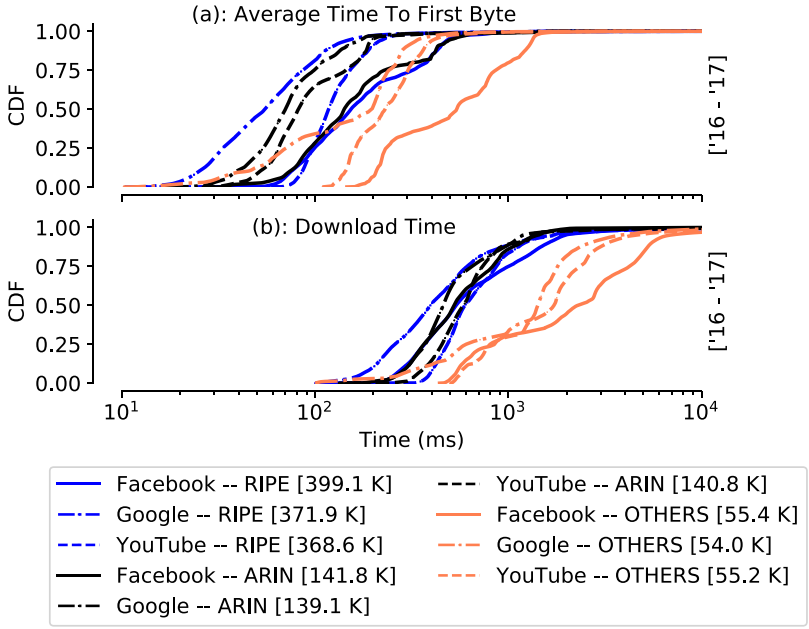

Figure 4.6. The CDF of the average TTFB and download time of the webpages across different origin ASes grouped by region. The average TTFB of Facebook is longer in the rest of the world than Europe and North America.

4.6 Web Latency by Region and Service Provider

To understand the web latency variation between regions and service providers, we dissected a year-long subset of our dataset (*i.e.,* 08/2016 to 07/2017). In this section, we present our analysis on the performance of the three webpages across the regions and the selected service providers.

By region. We grouped the origin ASes of the measurement probes by region as RIPE (Europe, Middle East and part of central Asia), ARIN (North America) and OTHERS (includes the rest of the world). Figure 4.6 shows the distribution of the average TTFB and download time of the webpages as observed from the three regions. As can be seen, all the webpages have different download performances in different regions. The probes hosted in regions other than ARIN and RIPE show worse performance for all webpages. A reason for this could be the absence of caches (see Section 4.3) and longer AS path lengths from the probes towards the websites' CDN.

By service provider. We analyzed the web latency of the webpages as observed from selected ISPs that host more than ten SamKnows probes. Most of the origin ASes in our dataset host up to five probes. Only three ASes, *i.e.,* Telecom Italia (TI), British Telecom (BT) and Comcast (CC) host

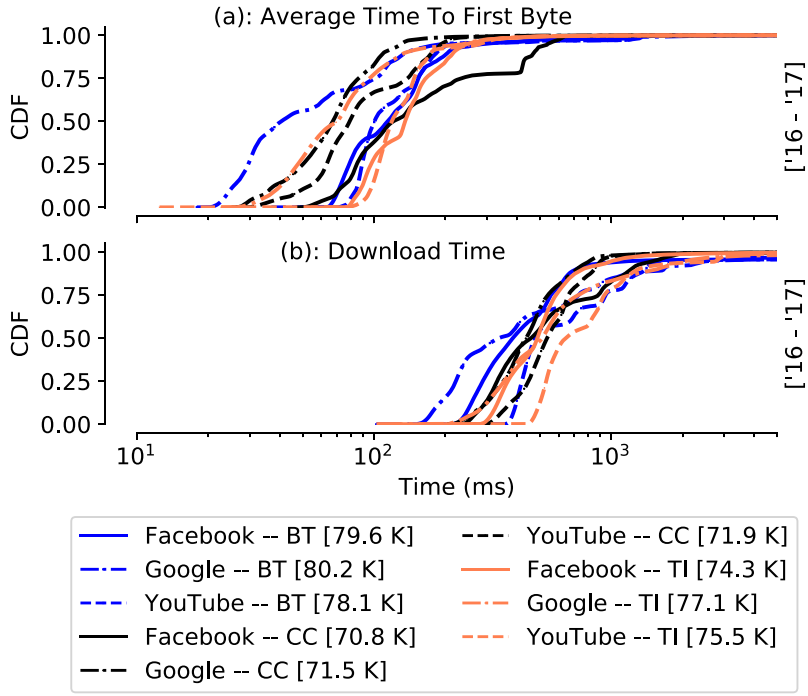

Figure 4.7. The CDF of the average TTFB and download time of the three webpages over selected origin ASes.

14, 13, and 13 probes, respectively. For the purpose of our analysis based on service provider, we focus only on these ISPs. Figure 4.7 shows the distributions of the average TTFB and the download time of the webpages for the three selected ASes. The result shows that Facebook exhibits a longer TTFB for 40% of the probes hosted in the CC network. A reason for this could be the longer AS path length to the Facebook CDN. Furthermore, YouTube had slower download performance for 70% of the probes hosted in the TI network. A previous study [190] in 2015 reported that Google has less (27) cache instances in Italy compared to the UK (67) and the USA (296). This could be a reason for the slower download performance of YouTube in the TI network.

4.7 Rendering Performance

To study the rendering performance of the webpages we used our measurement system, *WePR*, and performed a nine-month long (March 2015 to December 2015) measurement. We used three virtual machines (VMs) as rendering machines and one VM as a load balancer. We deployed the

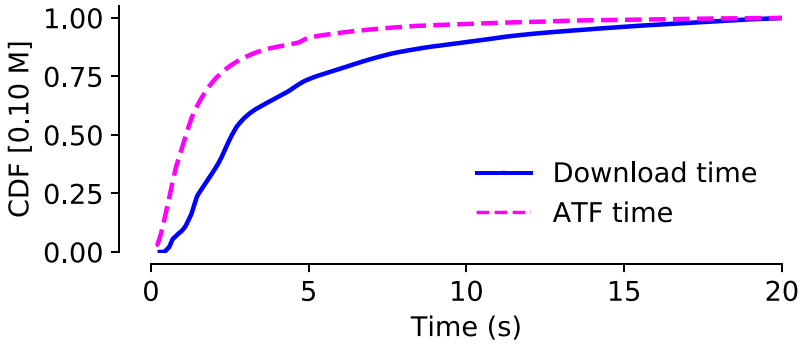

Figure 4.8. The download time and ATF time of the websites.

WebPerf test on 65 SamKnows probes located at different origin ASes and measured four websites. The websites were –

- `www.bbc.com`
- `www.ebay.com`
- `www.sina.com.cn`
- `www.reddit.com`

The websites a median object count of 62, 176, 131, and 32, respectively. The probes were set to measure these websites at four hour intervals. The websites were chosen from different categories to cover a range of content types and rendering behavior.

To approximate the QoE, we calculated the ATF time – the time to download and render the contents of a webpage within the above-the-fold area – of the websites using the pixel-wise approach (see Chapter 3.2). In most of the cases, the rendering behavior of the websites was that after the initial download period most of the website content was displayed almost instantly. Additional objects were also downloaded later, but they did not have a significant visual effect on the appearance of the above-the-fold area of the website. We also observed some cases in which the above-the-fold area of websites took longer to reach a stable state. The reason for these kinds of cases could be the presence of animated content and auto-play enabled video advertisements on the website. This type of content changes frequently and hinders the website from stabilizing.

Figure 4.8 shows the distribution of the download time and the ATF time of the four websites. As can be seen, the download time was significantly longer than the ATF time. The download time and the ATF time have a positive correlation (*i.e.,* a Pearson correlation of 0.41). To verify whether the ATF time was always shorter than the download time, we calculated

the difference between the download time and the ATF time for each measurement. The result shows that in 20% of the cases, the ATF time was longer than the download time. This means that even though the web objects are downloaded, it may take longer to be shown to the user. A possible reason for this could be the browsers rendering engine took longer to process and render the web contents. Note that in the measurements, we observed a heavy tail in the download time. We speculate that the heavy tail is due to third-party contents which may not have been optimized for the location of the probes. That is, the probes may have struggled to download these third-party contents although they do not have a visual effect in the appearance of the websites.

Moreover, we observed that the location of the content relative to the clients' geographical location had an impact on the rendering performance. For instance, `www.bbc.com` is hosted in the UK, while `www.sina.com.cn` is hosted in China. Due to the fact that majority of the probes were located in Europe, the effect of the sever location is clearly visible when we compare the rendering behavior of the two websites. That is, the ATF time of `www.sina.com.cn` shows a heavy tail distribution. This is because the website is likely not optimized for Europe.

We also explored how the throughput and latency (*i.e.,* the round trip latency of a single packet) affected the rendering performance of the websites. We used a `speedtest` and `ping` measurement that ran on each probe. These tests ran independently from the web rendering test. However, we took the hourly average from all tests for each probe and see how the results correlated. We found that there was no direct correlation between the throughput and the rendering performance of the website (*i.e.,* a Pearson correlation coefficient of -0.17). There was a weak correlation (*i.e.,* a Pearson correlation coefficient of 0.11) between the latency and the rendering performance of the websites.

4.8 Summary

This Chapter provides an analysis of web latency using 3.5-year-long dataset and QoE in fixed-line networks. We used *Webget* (see Section 3.3.1) and measured the DNS lookup time, the TTFB and download time of three popular web pages from geographically distributed locations for 3.5 years (2014 to mid-2017). Over the years, the DNS lookup time of the web pages improved over the course of our measurements. However, the download

time of the web pages did not show a clear general trend, as it depends on several factors including the presence of a catch, and web page complexity. The result also shows that an improvement in the broadband speed does not necessarily improve the web latency. Moreover, the web latency varies between regions and service providers. For instance, the web pages had a worse download performance in the regions other than ARIN and RIPE. We also observed a high latency from the probes hosted in the Comcast network towards to Facebook, and slower YouTube download performance from Telecom Italia. The reasons for these poor performances are the longer AS path length and the availability of caches.

The main takeaway from the longitudinal analysis of the three popular web pages is a number of factors such as web page complexity and the availability of caches affect the web page download performance. Because of the difference in availability of catches, the download performance for these web pages varies between regions and service providers.

We also compared the web page download performance and rendering performance of four web pages, using a nine-month long dataset. The result shows that the download time is longer than the ATF time of the web pages. The next Chapter presents a detailed analysis of the ATF time in cellular networks.

5. Web QoE in Cellular Networks

In this chapter, we discuss the main observations on the web QoE in cellular networks. Section 5.1 presents the dataset we collected. In section 5.2, we discuss the IP path length towards popular CDNs over fixed-line and cellular networks. Section 5.3 presents the analysis of the web QoE between different MNOs. Finally, section 5.4 presents the impact of mobility on the web QoE. This chapter is based on Publication IV.

5.1 Dataset

We deployed our web latency and rendering performance measurement tool – *WebLAR* (see Chapter 3.3) – on a European-wide mobile network measurement platform – *MONROE* (see Chapter 2.1.4) and conducted two (one week long each) measurement compaigns (May 19 - 26, 2018 and July 2 - 9, 2018). We measured the web performance from 128 vantage points (MONROE nodes) located in Norway and Sweden. The MONROE nodes were connected to the Internet through multi-homed (with one or more) LTE connectivity. Some nodes also had Ethernet and WiFi connectivity, which helps to compare the performance between LTE and WiFi networks. Nine of the nodes were connected to a Swedish operator and roaming in Norway. The nodes were set to execute the WebLAR experiment every six hours. We measured specific web pages of eight popular websites. The websites (along with the median number of objects per website) were –

- News websites
  - `http://www.bbc.com` (102)
  - `https://news.google.com` (84)
- Wiki websites
  - `https://en.wikipedia.org/wiki/Alan_Turing` (48)
  - `https://www.reddit.com` (120)
- Social media websites

  - `https://www.youtube.com` (56)
  - `https://www.facebook.com/places/Things-to-do-in-Paris-France/110774245616525` (91)

- General websites
  - `https://www.microsoft.com` (41)
  - `https://www.yahoo.com` (7)

The webpages were selected from different popular categories where user interaction is not required to show meaningful content. While we chose the webpages, we also took into consideration the design and the purpose of the websites. Note that, we were aware that these webpages have a mobile version, but we measured only the desktop version of the respective webpages. In the rest of this chapter, we refer to the webpages with the base name of their URL. After data cleaning by removing experiments that failed to report all the metrics (*e.g.,* due to browser timeout settings), we were left with $\sim$18K data points.

5.2 IP Path Lengths

To investigate the IP path length and the round trip latency in fixed-line and cellular networks, we ran a `traceroute` on 29 MONROE nodes located in Italy, Norway, Spain, and Sweden towards four popular CDNs. The nodes were connected to the Internet via both fixed-line (*e.g.,* academic network) and LTE connectivity. Figure 5.1 shows the IP path length to the four CDNs in the fixed-line and cellular networks. As can be seen, in the median case, the IP path length over the fixed-line and LTE networks were similar. In this plote, we showed only four out of the total of eight websites we measured. This is because the other four websites did not reveal the IP end points for web pages and we were not able to collect the IP path information.

5.3 Web QoE Across Different Operators

To quantify the QoE of the websites, we measured the ATF time of the websites using the two different approaches (see Chapter 3). Figure 5.2 shows the distribution of the ATF time of the websites between different operators. The top plot shows the ATF_b time (approximated using the browser heuristic approach), while the other three plots show the ATF_p time (*i.e.,* the ATF time approximated using the pixel-wise comparison

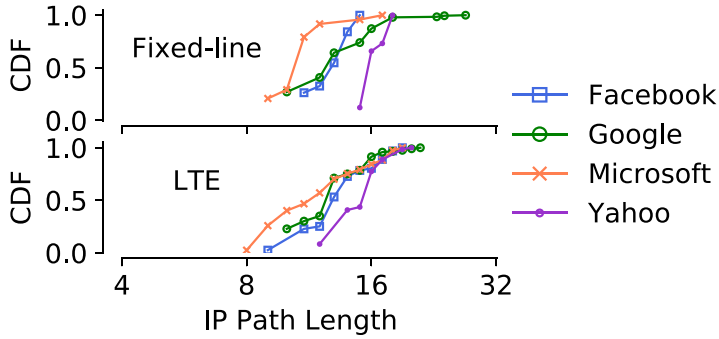

Figure 5.1. The distribution of the IP path length towards the four websites over fixed-line and LTE networks.

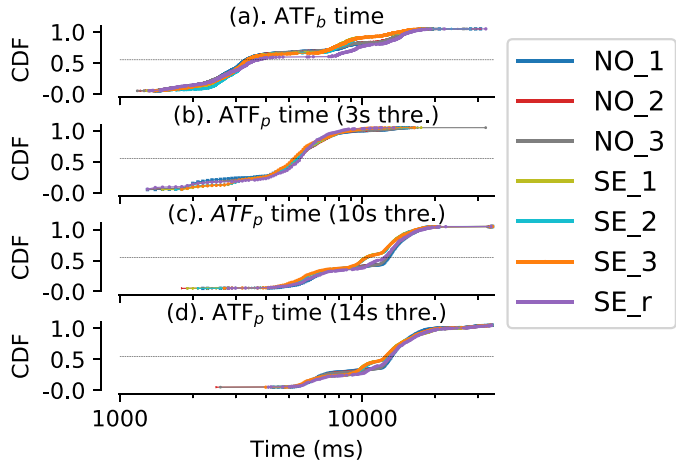

Figure 5.2. The distribution of the ATF time approximated using the two approaches. The Norwegian and Swedish operators are labeled NO_o and SE_o, respectively, where $o \in 1, 2, 3$. SE_r represents a Swedish operator roaming in Norway.

approach with different website stabilizing thresholds). We can observe a long tail distribution in the ATF$_b$ time (*i.e.,* the ATF time approximated using the pixel-wise comparison approach). This is due to the higher number of overlapping images in the above-the-fold area of BBC and Facebook.

As can be seen, in the median case, the ATF$_b$ time is shorter than the ATF$_p$ time with a three-second threshold. This hints that, in cellular networks, the above-the-fold area of the websites reach a stable state within a three-second threshold. As such, going forward, we consider only the three-second threshold when approximating the ATF time using the pixel-wise comparison approach.

When we compared the difference in the ATF time of the websites between different operators, we saw a small variation within most of the

operators. The difference was from 100 ms to 300 ms, in the median case. However, the difference between the roaming operator (SE_r) and the other operators was relatively higher. In the median case, the difference in the ATF_b time can be 400 ms and the ATF_p time can be up to 4200 ms. To investigate the significance of the difference in the ATF time across the MNOs, we applied a Kolmogorov-Smirnov test [191]. The test result shows a smaller p-value (below 0.05), which confirms that there is a difference between the ATF_b time of the websites for the different MNOs. However, we found a higher p-value (0.75) on the ATF_b time of the websites between the two Swedish operators (SE_1 and SE_3). This indicates that the two MNOs give a similar browsing experience.

We also looked at the impact of roaming on the QoE. As can be seen from Figure 5.2, the Swedish operator roaming in Norway has an almost similar QoE to the native Swedish operators. This indicates that the home-routed roaming [192] configuration does not have much impact on the QoE for users who travel a short distance.

Furthermore, we analyzed the rendering behavior of each website by focusing on the ATF time, approximated using the two approaches and the PLT – the time until all objects in the webpage are loaded and the `onLoad` event fires. Figure 5.3 shows the ATF time approximated using the two approaches and the PLT of the websites. As can be seen, the ATF_b time is shorter for websites that have fewer images in the above-the-fold area, such as Microsoft. For websites like Facebook which have multiple overlapping images in the above-the-fold area, the ATF_p time is shorter. For these kinds of websites the ATF_b time is longer. This is because the websites take a longer time to download the images in the above-the-fold area, but the images do not change the websites visual appearance unless the user takes an action (*e.g.,* clicking on a sliding button). We manually analyzed the number of objects of each website and observed that the number of images in the above-the-fold area varied between the websites we measured.

The ATF time of a website approximated using the two methods is supposed to be similar. However, as we observed in our results, the websites may have a different ATF time when approximated using the two approaches. For websites like Facebook that have multiple overlapping images in the above-the-fold area, the browser heuristic approach overestimates the ATF time. That is, a delay in the download and rendering of images that are not visible in the front does not affect the appearance

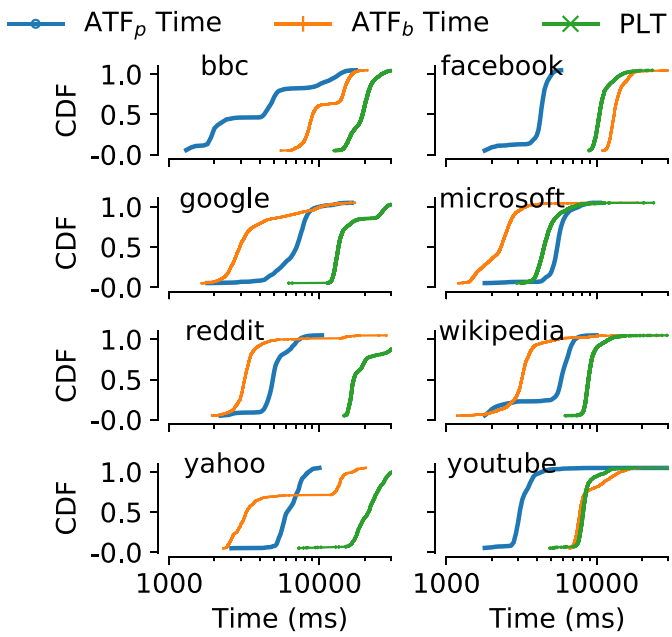

Figure 5.3. The distribution of the ATF time approximated using the two approaches.

of the websites. Hence, it does not have an impact on the user browsing experience. In contrast, websites such as Microsoft that have animated contents in the above-the-fold area may not stabilize. In such cases, the pixel-wise comparison approach may not work well to approximate the ATF time.

To make a concrete example, let us compare the ATF time of the websites when approximated using the two approaches. In the case of Facebook, the ATF$_b$ time is longer than the PLT which is also longer than the ATF$_p$ time. This is due to the fact that Facebook downloads contents asynchronously and the webpage has overlapping images in the above-the-fold area. When we consider the case of Microsoft, although it has a lower number of images in the above-the-fold area, the visual look of the webpage seems to be manipulated using CSS and JavaScript and it has animated contents. As a result, the pixel-wise comparison approach yields a longer ATF time. Therefore, the design and the types of the website play an important role in deciding which methodology to use for approximating the ATF time. Moreover, because of the design patterns adopted by some websites, objects may be downloaded asynchronously and the TCP connection may not be closed even if the `onLoad` event fires. In such cases, the PLT of the website may be shorter than the ATF$_b$ time.

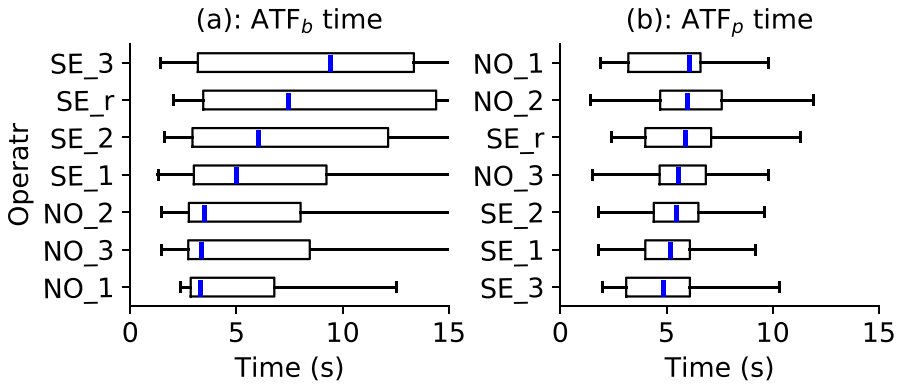

Figure 5.4. The distribution of the ATF time approximated using the two approaches for different operators under a mobility scenario.

An observation from Figure 5.3 is that, the ATF time of some websites like BBC, Yahoo and Wikipedia show a bimodal distribution. This is due to the fact that Yahoo shows variation in the ATF time across different countries. That is, accessing it using the Norwegian MNOs, Yahoo takes longer to show the contents in the above-the-fold area. A reason for this could be the absence of Yahoo's cache close to the Norwegian ISPs. However, for the other two websites, we observed the bimodal behavior across all operators.

5.4 Web QoE Under Mobility Conditions

To understand the impact of mobility on web QoE, we studied the ATF time of the websites as measured from the MONROE nodes under mobility conditions. Throughout the course of the measurement, more than half of the nodes were deployed on trains, buses and trucks. Figure 5.4 shows the distribution of the ATF time of the websites for different MNOs under mobility conditions. As can be seen, the ATF time of the websites measured from nodes deployed on trains and buses is similar to that of the nodes deployed in homes and offices. However, the variation in the ATF time between different MNOs is relatively higher under the mobility scenario.

Figure 5.5 shows the distribution of the ATF time and PLT of websites under different mobility scenarios. As can be seen, the websites have a similar PLT in stationary and mobility conditions. However, the ATF time of some websites is longer when the nodes are moving. For instance, in the median case, the ATF time for Microsoft, Yahoo, Reddit, and Facebook were 0.3 to 1 second longer in the mobility scenario than in stationary

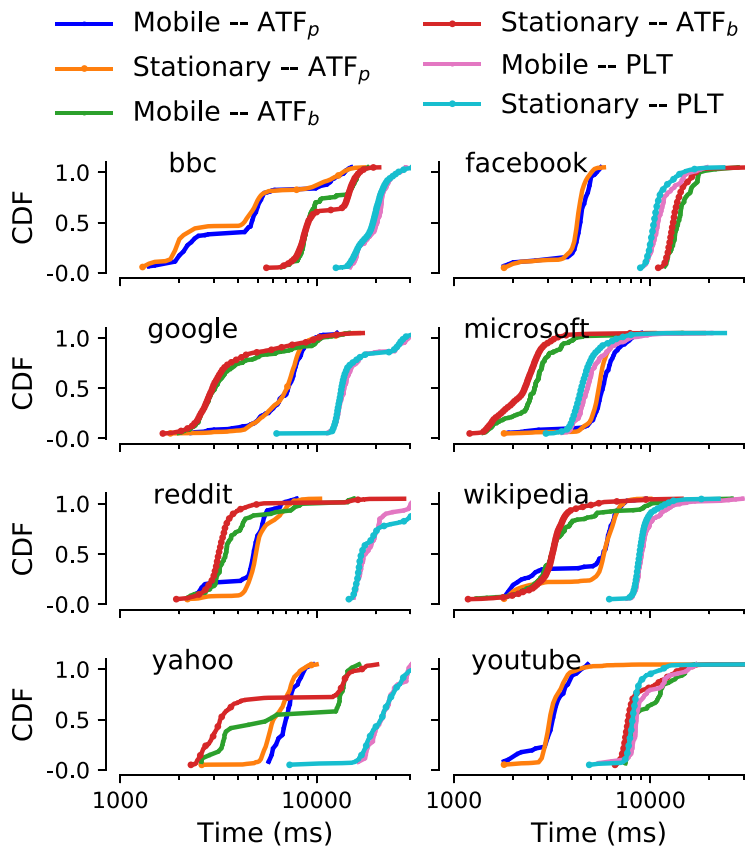

Figure 5.5. The distribution of the ATF time and PLT of the websites under different mobility conditions.

condition.

One observation from this result is that Yahoo has different ATF times from both stationary and mobile nodes. That is, in about 60% and 40% of the mobile nodes and stationary nodes, respectively, showed a drastic change in the ATF time. The cause for this variation is that the nodes connected to the Norwegian operators took longer to display the contents in the above-the-fold area. A possible reason for this could be a longer IP path length towards the Yahoo content location from the nodes located in Norway. Using a `traceroute` measurement, we found that in order to reach Yahoo's web server, the clients needed to traverse up to 20 and 16 IP hops from Norwegian and Swedish MNOs, respectively.

5.5 Summary

This Chapter presents the results describing the web QoE in cellular networks collected from geographically distributed vantage points. We measured the ATF time of eight popular websites from nodes connected to the LTE network. The results show that the difference in the ATF time of the websites varied slightly between different operators. Moreover, the result show that in cellular networks, the websites finish downloading and rendering the contents in the current viewport in three seconds. The ATF time of the websites approximated using the two methods may vary depending on the design and type of the websites. We also witnessed that home-routed roaming does not have a significant impact on the QoE for users who travel shorter distance.

The impact of mobility on the website download performance was limited. However, for some websites we observed that the ATF time was longer when the users were moving. In the next Chapter, we focus on the ATF time approximated using the browser heuristics and show how it is related to the actual user subjective rating.

6. Modeling Web Quality of Experience

In the previous chapters, we presented the methods and tools to automatically measure the web QoS metics and approximate ATF time. In this chapter, we investigate how the web QoS metrics relate to the user browsing QoE. Section 6.1 presents the dataset that we use for our analysis. Section 6.2 discusses web QoE modeling from the web QoS metrics using mathematical models. Finally, in section 6.3, we present Machine Learning (ML) approaches to modeling web browsing QoE. The contents of this chapter is based on Publication III.

6.1 Dataset

In this section, we discuss the measurement we conducted to collect web browsing dataset annotated with user openion. Then, we describe the data sanitization steps and how we calculate MOS for the web page we measured.

6.1.1 Experiment

To evaluate the impact of application-level QoS on the end-user QoE, we extended a previous experiment by Bocchi *et al.* [193] on measuring web QoE. To have full control of the experiment (*e.g.,* the network configuration), we downloaded the top 100 most popular websites in France (from Alexa list [194]) and hosted on six local servers. During the download process, we also recorded metadata such as network delay and packet loss. The servers equipped with a quad-core processor, 4 GB of RAM, and two Gigabit network cards. The servers run Ubuntu 14.04 with Apache HTTP Server 2.4.18. The Apache server run with the default configuration, and HTTP/2 and SSL (with self-signed certificate added) modules were enabled.

We conducted a controlled measurement involving 241 volunteers (with no demographic exclusion). The volunteers were asked to watch when the webpages were loaded in Google Chrome browser and provide their subjective rating using the Absolute Category Rating (ACR) [195] – *i.e.,* from 1 (bad) to 5 (excellent). The volunteers were provided a PC running Linux Mint 17.3 equipped with a set of scripts that orchestrate the experiment. The scripts set up the experiment with different network scenarios, open Google Chrome browser in full-screen mode, collect user's score and objective metrics and send the results to a central server. The steps in the experiment are as follows. First, a GUI with a list of websites available for the study is opened. Then, the user selects a website from the list and observes while the website is loaded in Google Chrome. At the end, the user enters a subjective rating for the website and watches again while the same website is loading but now served with a different protocol (*i.e.,* HTTP/1 or HTTP/2). When a website loaded for the first time, the protocols were chosen randomly and in the next step the other protocol was used. The users were not aware of which protocol was used when a website was loaded. In the experiment, the volunteers were exposed to randomly choose one among three network scenarios. The different network scenarios are: (a) The objects are distributed as observed originally but artificial latency (RTT of 0 ms, 20 ms, 50 ms, and 100 ms) and packet loss (0%, 1%, 2%) were forcefully introduced in the six servers. (b) The objects were distributed as originally observed and no artificial latency or packet were loss introduced. (c) All the objects were hosted by a single server. Artificial latency (RTT of 0 ms, 20 ms, 50 ms, and 100 ms) but no loss were introduced. Furthermore, the scripts also were set to collect the network events in the form of HAR file for later analysis. The experiment workflow and the detailed configuration setup are available in [193].

We collected a total of 8,689 web browsing sessions that were based on 25 unique webpages of the aforementioned list of websites (broken webpages and langing pages were excluded). The webpages have variations in their page complexity, *i.e.,* the number of objects (24 to 212), the size of webpages (0.43 to 2.88 MB), and the number of servers (1 to 30).

6.1.2 Dataset Cleaning

Since our experiment relied on volunteers, we applied basic data sanitization techniques. First, we removed users who did not complete at least 10 browsing sessions and provide feedback. With this, we keep 224 subjects

out of the original 241 volunteers, and 8,568 out of 8,689 user ratings. Finally, we restricted our analysis to only 12 webpages out of the original 25 webpages comprising a significant number of reviews and experiment conditions, which left us with 3,400 user ratings.

Mean Opinion Score (MOS). Before calculating the MOS for each web page, we grouped the users' ratings based on the distribution of the characteristics of the input QoS metrics such as PLT. We placed the user ratings into six groups; that is, at every 20^{th} percentile of the metric until the 80^{th} percentile, and at every 10^{th} percentile for the remaining population. Then, we calculated the MOS value for a webpage by averaging the opinion scores of a specific group. In doing so, we reduced the sampling bias of the MOS calculation due to variation in the opinion score as a consequence of differences in the network configuration, web transfer protocols, etc.

6.2 Mathematical Models

In this section, we used the annotated dataset (6.1) and applied mathematical models to approximate the user QoE from the QoS metrics. To map the application QoS metrics to the user QoE, we used three mapping functions, namely a linear, a logarithmic, and an exponential function. As discussed in Section 2.4.4, the rationale for using a logarithmic function was due to the fact that the input QoS parameters such as the PLT have a logarithmic relation with the resulting QoE, as is the case in the Weber-Fechner law. Moreover, the exponential function follows the IQX hypothesis, in which the changes in the QoE depend on the current level QoE. We also evaluated the linear mapping function as many studies still use a direct comparison between the statistics, such as PLT to user-perceived QoE. As such, we believe that evaluating the three mapping functions in our dataset was important to lay a foundation for modeling the user browsing QoE.

To assess how well a function $f(\cdot)$ applied to a QoS metric x correlates with the MOS, we consider the nine metrics: DOM, ATF, PLT, BI^{ATF}, BI^{PLT}, OI^{ATF}, and OI^{PLT}. These metrics are presented in Section 2.4.3, and we briefly described below.

- The first class of mertics we considered are time-instance metrics including the time to load the DOM structure, ATF time (*i.e.,* the time to load the last visible image or other multimedia object), and PLT (*i.e.,* the time trigger the `onLoad` event).

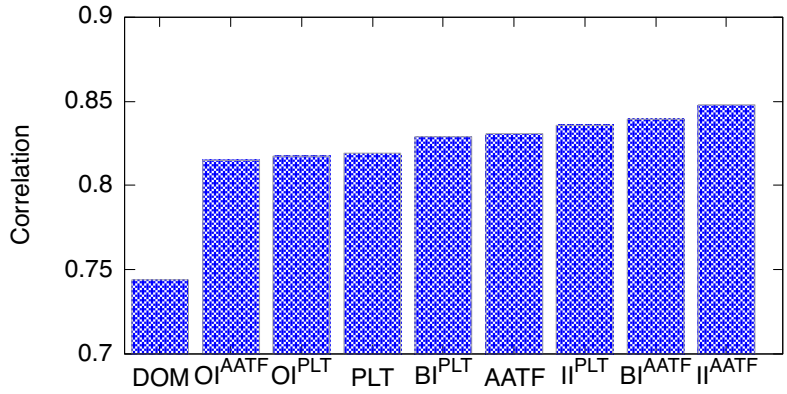

Figure 6.1. The impact of nine selected QoS metrics on the QoE using the IQX model.

- The second category of metrics are time-integral metrics such as ByteIndex (BI) and ObjectIndex (OI) with ATF time and PLT cut off points. The cutoff function $x(t)$ represents the percentage of bytes downloaded at time t for the BI and the percentage of objects downloaded at time t for OI.

- Finally, to identify the contribution of images in the rendering behavior of the webpages, we further define Image Index (II) with the ATF time and PLT cutoff point. The Image Index represents the size of the objects of the image class within a given time horizon.

We applied all the selected nine QoS metrics to the three mapping functions and computed the correlation with the user QoE. Figure 6.1 shows the correlation of nine selected QoS metrics with the user QoE using the exponential mapping function (based on the IQX hypothesis). As can be seen, the DOM shows the weakest correlation. All the other metrics show strong correlations (Pearson correlation > 0.8) with the MOS. The results also confirm that counting the bytes (BI), more specifically the image bytes (II), is more valuable than counting the number of images (OI). That is, the metric with the strongest correlation (0.85) is the image index narrowing with the ATF time-horizon (II^{AATF}). The commonly used QoE metric – PLT – has a correlation of 0.81. This confirms the soundness of using the ATF time to approximate the user-perceived web page loading time.

To compare which mapping function is better to approximate the user QoE, we selected the most widely used metric (PLT), the metric that exhibited worst (DOM) and the best (II^{ATF}) correlations with the user QoE. Figure 6.2 shows the comparison of how different mapping functions correlate with the QoE for a selected subset of the QoS metrics. As can

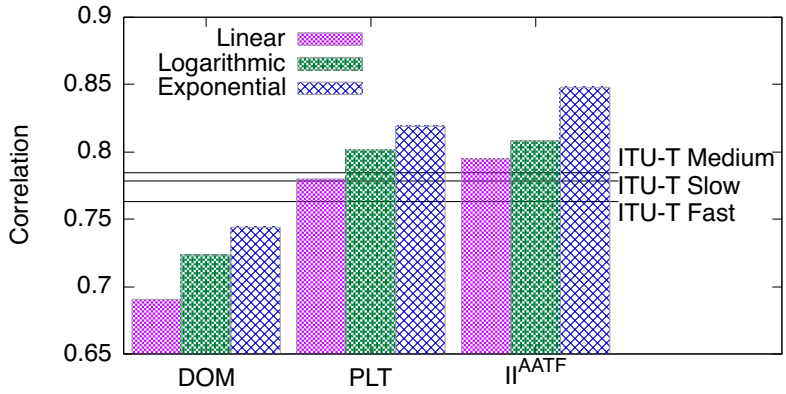

Figure 6.2. The impact of three selected metrics on the expert mapping functions. The bar show correlation of the metric x applied on the mapping function $f(\cdot)$ and the QoE. The lines show the comparison of the results from the reference obtained by applying the default ITU-T models for slow/medium/fast network condition using the PLT metric.

be seen, across all three metrics in our dataset, the exponential mapping function is better than the logarithmic one, which in turn is better that a simple linear mapping function to estimate the user QoE. Overall, the ATF time yields the strongest correlation with the MOS across all mapping functions.

Furthermore, we also compared the results with the reference obtained by applying the ITU-T models [142] for slow, medium, fast network conditions (see Chapter 2.4.4) using the PLT metric (the lines in Figure 6.2). The medium network condition shows the strongest correlation with the QoE in our dataset. The reason for this could be related to the user's expectation in network conditions as the experimental network conditions mirror that of Internet web access.

6.3 Machine Learning Models

To learn the regression model that predicts the user QoE, we used state-of-the-art machine learning (ML) techniques and evaluated the prediction performance. In this work, we evaluated three ML algorithms: Support Vector Regression (SVR) [196], Classification And Regression Tree (CART) [197], and AdaBoost with CART (BOOST) [198]. Unlike the expert models which take a scalar metric, functions $f(\cdot)$ learned from a regression model map a vector of metrics to MOS. The algorithms were implemented using the scikit-learn Python module [199].

6.3.1 Parameter Tuning

To obtain accurate modeling results, finding the optimal hyperparameter is an essential step for learning algorithms. Setting a proper hyperparameter plays an important role in controlling the flexibility of the model to fitting the data, as well as the robustness, and prediction accuracy of the models. These values are sensitive to small changes in the learning rate and can lead to a significant change in the accuracy of the model. These parameters help reduce the learning errors, and help to find a balance between the bias and variance that prevents the model from overfitting or underfitting. Among the different tuning methods, in our work, we used a grid search to set the hyper-parameters of the machine learning algorithms. For each algorithm, we selected the hyperparameter combinations described below. For the other parameters of the algorithms, we used the default settings provided in the scikit-learn Python [199] implementation.

SVR. This depends on hyperparameters such as the thickness of insensitive zone ϵ, a penalty factor C, and a "radial bias function (RBF)" kernel parameter γ [200]. The parameter C describes the tolerance of the errors during the training process (*i.e.,* $C = 1.0$ means no tolerance for errors and the penalty is more important for the regression; whereas $C = 0.0$ gives an extreme tolerance for errors). We set the hyper-parameters to $\epsilon \in [10^{-2}, 1]$, $\gamma \in [10^{-3}, 10]$ and $C \in [1, 10^4]$.

CART. Decision trees mainly depend on hyperparameters such as the maximum tree depth and the minimum number of samples per leaf. In our work, we set the minimum number of samples per leaf to $\in [1, 10]$, and the tree depth to $\in [1, 10]$.

BOOST. Boosting works by combining less accurate models to create a mode with high accuracy, we built our boosting model on top of the CART model. As such, we used the same settings as the CART algorithm and set the number of boosted trees to $\in [10, 10^3]$.

After tuning the hyperparameters using the grid optimization technique, the optimal outputs were $\epsilon = 0.3$, $\gamma = 10^{-3}$, and $C = 10^4$ for the SVR. For both the CART and BOOST algorithms, the optimal samples per tree were 4 with a tree depth of 2. In addition, the hyperparameter tuning suggested 10^2 boosted trees for the BOOST algorithm.

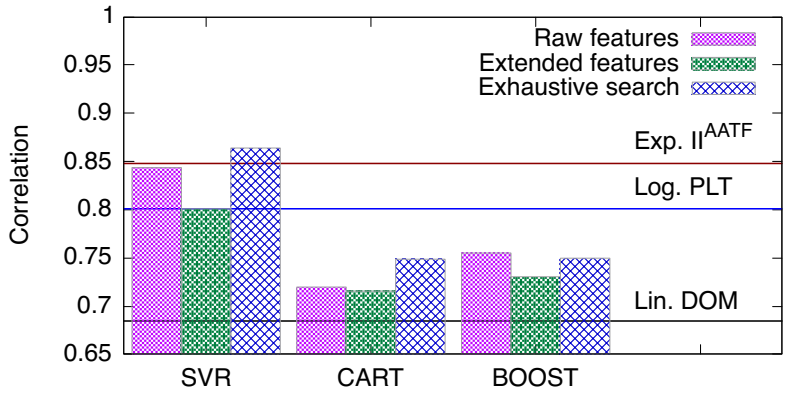

Figure 6.3. The correlation between MOS and machine learning algorithms using different feature sets against reference expert models.

6.3.2 Feature Selection

In the process of building the prediction models, reducing the number of input variables is essential. We followed three strategies to select the relevant attributes from our dataset.

1. As a baseline set of features, we used the nine raw metrics defined in Section 6.2.
2. We considered a set of 27 extended features, created by combining the output of the three expert models (*i.e.,* linear, logarithmic and exponential) with the nine raw metrics.
3. Lastly, we performed an exhaustive search of feature subsets from the extended set, to select a combination of features that reduced the RMSE. The combinations included few features (three to five out of nine raw metrics) that varied between different algorithms. The combination sets consistently contained II^{PLT} for all the algorithms. Moreover, ATF and II^{ATF} also found in the combination set for at least for two algorithms.

6.3.3 Results

To assess the performance of the machine learning models, we evaluated the predictors using leave-one-out cross-validation. Moreover, we calculated the Pearson correlation and RMSE for each algorithm. Figure 6.3 shows the Pearson correlation between MOS and the ML model for the different algorithms and feature selection approaches. As can be seen, for the

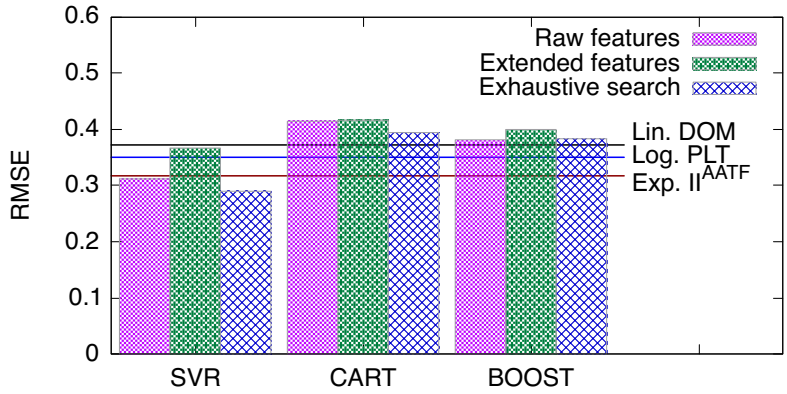

Figure 6.4. The RMSE of the prediction algorithms using different feature sets against reference expert models.

correlations (where higher is better) the SVR algorithm outperforms both the CART and BOOST algorithms. Moreover, BOOST shows an advantage over CART.

Figure 6.4 shows the prediction errors (expressed in terms of the RMSE) for the different modeling algorithms and feature selection approaches. As shown in the figure, the SVR algorithm has smaller prediction errors (*i.e.,* lower RMSE values indicate a better prediction performance) than both the CART and BOOST algorithms. Moreover, the BOOST algorithm shows a smaller RMSE than CART, which confirms our claim above.

Furthermore, we compared the performance of the machine learning results to the expert model results. Figure 6.3 shows a comparison of the Pearson correlation between the user MOS and the predicted MOS by using the machine learning and the three mathematical models (*i.e.,* the best (exponential with II^{ATF}), the commonly used (logarithmic with PLT), and the worst (linear with DOM)). Figure 6.4 also shows a comparison of the RMSE of the MOS estimation by the machine learning models and the three expert models. As can be seen, the SVR result is as good as the best expert model (*i.e.,* the exponential model with II^{ATF}). Another observation is that the CART and the BOOST results were not as good as the commonly used expert model (*i.e.,* logarithmic with PLT). Moreover, the results show that an exhaustive search for feature selection yields an advantage over the other feature selection strategies. Additionally, we observed that using the raw features as an input for the machine learning algorithms yielded a better prediction performance than using the extended features

The main takeaway is that among the different machine learning algorithms we used to predict the user experience, the SVR result shows a

strong positive correlation with the MOS and the smallest prediction error. Our results show that applying the SVR algorithm results in a comparable prediction performance to the best expert mathematical model. However, using extended features created by combining the outputs from the expert modes did not result in a better prediction performance for the three algorithms. Instead, applying an exhaustive search over the extended features to find the best combination of features yields a better QoE prediction.

6.4 Summary

This chapter discusses modeling web QoE from the QoS metrics. We conducted a controlled experiment and collected more than 8,000 web browsing sessions annotated with user raring based on 25 unique websites. After performing data sanitation (*e.g.,* removing users who did not provide a sufficient number of ratings), we left 3400 user ratings for 12 unique websites. We used state the art expert and machine learning models to predict the user MOS from the QoS metrics. From the exert mapping functions, we tested the linear, logarithmic, and exponential relation between the QoS and MOS. After evaluating the three mapping functions with nine different metrics, we found that exponential mapping with the II^{ATF} metric showed a higher correlation with the user MOS. This suggests that counting the images and narrowing them down to the ATF time-horizon is more important in approximating the user QoE.

We also evaluated the prediction performance of Support Vector Regression, Classification And Regression Tree, and AdaBoost with CART machine learning algorithms using our labeled dataset. We applied a grid search optimization technique to tune hyperparameters of the algorithms. We used different feature sets as an input for these algorithms and computed the prediction error and correlation with the MOS. The results show the SVR algorithm yielded a better prediction than the CART and BOOST algorithms. The SVR results are compare with the best expert mode (*i.e.,* an exponential model with II^{ATF}). In the next chapter, we conclude the thesis by discussing the implications, reproducibility, limitations, and future work.

7. Conclusions

The web is one of the dominant applications on the Internet. The web ecosystem has been evolving over the last three decades. The web consumes a significant traffic share on the Internet. There is also a huge demand for fast and reliable browsing from the end-user perspective. To satisfy the users' needs, content and service providers have been proposing and implementing different optimization techniques and advancements. The advancements include various kinds of web building technology, content provisioning, and transport and application protocols.

Understanding the user browsing experience is important for web designers, content, and service providers. A bad browsing experience leads to customer churn and has an impact on business revenue. The content and service providers usually run tests (*e.g.,* latency measurements) on their own premises (for instance, on the backbone network) and try to measure the performance and understand the service quality that the user gets. However, these kinds of measurements which are conducted on the providers' side do not give a holistic view of the network performance. In addition, the user experience is subjective by nature and several factors such as the context, the transport network, and the user device can affect it. As such, a measurement taken from the users' premises is crucial to understand the users' perceived QoE.

Due to the limitations of subjective QoE assessments, objective metrics have been used to measure the user browsing experience. The most widely used objective metrics to measure the web QoE have been latency-related metrics such as the Document Object Model (DOM) time, Time To First Byte (TTFB), and the Page Load Time (PLT). These metrics measure the occurrence of a particular event in the waterfall of the browsing session and do not adequately measure the user perceivable browsing experience. As such, other objective metrics that better approximate the web QoE have

been proposed. The ATF time and Speed Index are the prominent metrics in this category. These metrics integrate several events that happen in a browsing session.

In this thesis, we proposed and designed a measurement system and tools to estimate the ATF time which better approximates the web QoE. In Chapter 2 we reviewed the necessary background on the foundation of Internet measurement and the platforms available that we could leverage for our work. We also discussed the evolution of the web, web performance, and factors affecting web performance. Furthermore, we presented web QoE and the QoE to map the QoS to the user perceivable experience. After laying this foundation, in the rest of the thesis, we presented the metrics and methods, and the measurement system and tools. Finally, we discussed models that map the objective QoS metrics to the user experience.

We proposed two methods to estimate the ATF time. One way is to use a pixel-wise comparison of the current viewport of the browser for a certain period, and the other way is to utilize resource timing API information. We implemented both methods. The measurement system and the tools can be used to measure the browsing experience on a large scale. We conducted longitudinal web latency measurements towards three popular web pages from 182 vantage points distributed across the globe. Our measurement results show the DNS lookup times towards popular CDNs improved over time. Also, the availability of caches within the ISPs network reduce the IP path length and this improves latency when accessing popular contents. Moreover, our web QoE measurements in cellular networks show that the ATF time websites have small variations with different ISPs. Our results show that mobility has a limited impact on the overall download performance of the websites, but for some websites the ATF time was longer when the nodes were moving. Further, we performed a controlled experiment in which we asked volunteers to browse a set of websites and provide their subjective ratings. Using this dataset, we modeled the user browsing experience from the QoS metrics. Our results show that an exponential model with counting the number of images under the ATF area better approximates the user QoE. Moreover, in our dataset the Support Vector Regression results achieved a higher prediction accuracy than the Classification And Regression Tree and AdaBoost with CART algorithms.

Our longitudinal measurement covers limited in number but popular web pages that are optimized and have higher number caches across the

world. The analysis of the dataset from this measurement gives insights how the web latency changes over time and between service providers and regions. Futhermore, our measurement over cellular networks covers eight websites that helps in understanding the web experience over cellular networks. The analysis of the datasets from these measurements has different implications in service and network management, which we discuss in the next section. In the remainder of this Chapter, we discuss the implications, reproducibility, limitation of the work, and future research directions.

7.1 Implications

The software and the results of this thesis will help web designers, content, and service providers to better design and manage infrastructures to improve the end-user experience. The community can leverage our ATF time estimation methodologies to better approximate the web QoE on a large scale. Moreover, our results help determine the optimal content locations to reduce web latency. Content and service providers can use our results to better manage their traffic towards popular web pages. This quantification encourages service providers to provision caches within their network so as to improve the user QoE.

ISPs can use our measurement system and tools to continuously monitor the user browsing experience in their customer base. ISPs can deploy small measurement boxes at the customers' home gateway, and execute our `WebLAR` measurement tool to monitor the web QoE from the customers' location. Collecting measurement data from the users' premises helps identify the root causes when bad browsing experiences happen. Vantage point distribution within the network helps the ISP identify whether a quality degradation for a user is due to a problem in the shared part of the network, or unique to a single user line, or in the home network, or a problem with the over-the-top service. Furthermore, ISPs can complement the measurements that they collect using our tools with passive traffic measurement that they can capture in their backbone network to understand bottlenecks and to better manage web traffic towards popular CDNs. The ISP can also use our measurement results for better capacity planning and network design.

7.2 Reproducibility

All the research work carried out as part of this thesis was done with the goal of scientific validity. As such, all the source code and other relevant test suites of the measurement system and tools (*WePR*, *WebLAR*, and *Approximate ATF*) have been open-sourced for the use of the scientific community [16, 15, 17]. Moreover, to encourage reproducibility, the datasets along with the processing scripts are also publicly available.

7.3 Limitations and Future Perspectives

The focus area of this thesis was measuring and inferring web QoE using active network measurements. The three-second website stabilizing threshold used in the pixel-wise comparison approach to estimate the ATF time does not consider web pages that have auto-play enabled video contents. Web pages that have auto-play enabled video need special but not trivial consideration to set the stabilizing threshold. The resource timing API does not provide complete information for some objects in the web pages. For instance, due to security reasons, browser APIs do not give timing information about cross-referenced objects unless it is explicitly enabled by the owners. Besides, we acknowledge that sometimes the resource timing information that the APIs provide may not accurate. Exploring different sources of timing information was beyond the scope of this work.

The implementation of the playback module of the `WePR` system does not support HTTPS and WebSocket requests. The content of a website may change depending on the location the request is sent from. In our `WePR` measurement setup, the parsing server was located at one vantage point only, which skews our view of content delivery. Our measurements cover a small set of popular websites. In our longitudinal measurement we did not cover requests over HTTP/2 or QUIC.

As a future research direction, this thesis could be improved in the following areas.

- Protocols such as HTTP/2 and QUIC have been under standardization and their deployment is steadily increasing. Our measurement system and tools do not support recent protocols that are aimed at improving web performance. Implementing support for the QUIC protocol in the measurement tools, and conducting measurements over QUIC have been left for future work.

- In the current design of the Intenet, web sessions requre strict reliability. However, all the contents of a web page such as images and videos may not require strict reliability to give a better user experience. As such, in the delivery of the web contnets, classify the web objects in to contents that require stirct (*e.g.,* Scripts, Stylesheets, and HTML files) and partial (*e.g.,* videos and images) reliability could give better end-user experience. That is, all the web requests and the contnets that require strict reliability is delivered over TCP and contents that does not require strict reliability could be delivered over UDP. One future vision for this thesis is understanding the impact of partial reliability on web browsing QoE.
- The proposed pixel-wise comparison method to estimate the ATF time could be improved to accommodate video and animated contents of the web page. One way to improve this could be considering the downloaded content checksum at different points in the waterfall of the web download session. This is considered beyond the scope of this thesis.
- Accessing 3D materials and augmented reality over the web is becoming common. These contents have different requirements than traditional web objects, for example, requiring higher processing devices and browser plugins. As a result of this, the web QoE metrics used for assessing the normal web pages are not sufficient to assess the user experience of the 3D web and augmented web. Therefore, new metrics and measurement methods are necessary to understand the user experience while browsing a 3D web or augmented web. This could be one potential future research direction.

To conclude, this thesis presented metrics, methodology and tools to approximate the web QoE using active network measurement. Along with the analysis of web measurement dataset, the thesis presented models to predict the web QoE from application level QoS metrics.

References

[1] CERN. A short history of the Web. https://home.cern/science/computing/birth-web/short-history-web. Retrieved on April 11, 2020.

[2] Tim Berners-Lee and Robert Cailliau. WorldWideWeb: Proposal for a HyperText Project. https://cds.cern.ch/record/2639699/files/Proposal_Nov-1990.pdf. Retrieved on Mar 18, 2020.

[3] Sandvine. The Global Internet Phenomena Report. https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf. Accessed on Aug. 15, 2019.

[4] Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. Tracking the Evolution of Web Traffic: 1995-2003. In *11th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2003) October 2003, Orlando, FL, USA*, pages 16–25. IEEE Computer Society, 2003.

[5] Tim Bray. Measuring the Web. *World Wide Web Journal*, 1(3), 1996.

[6] Troy A. Johnson and Patrick Seeling. Desktop and Mobile Web Page Comparison: Characteristics, Trends, and Implications. *IEEE Communications Magazine*, 52(9):144–151, 2014.

[7] Enrico Bocchi, Luca De Cicco, and Dario Rossi. Measuring the Quality of Experience of Web users. *Computer Communication Review*, 46(4):8–13, 2016.

[8] International Telecommunication Union. ITU-T Recommendation P.800: Methods for subjective determination of transmission quality. Technical report, 1996.

[9] Tobias Hoßfeld, Poul E. Heegaard, Martín Varela, and Sebastian Möller. QoE beyond the MOS: an in-depth look at QoE via better metrics and their relation to MOS. *Quality and User Experience*, 1(1):2, Sep 2016.

[10] Flavia Salutari, Diego N. da Hora, Gilles Dubuc, and Dario Rossi. Analyzing Wikipedia Users' Perceived Quality of Experience: A Large-Scale Study. *IEEE Trans. Netw. Serv. Manag.*, 17(2):1082–1095, 2020.

[11] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-fold Metrics. In *Passive and Active*

Measurement - 19th International Conference, PAM 2018, Berlin, Germany, March 26-27, 2018, Proceedings, pages 31–43, 2018.

[12] Tobias Hoßfeld, Florian Metzger, and Dario Rossi. Speed Index: Relating the Industrial Standard for User Perceived Web Performance to web QoE. In *Tenth International Conference on Quality of Multimedia Experience, QoMEX 2018, Cagliari, Italy, May 29 - June 1, 2018*, pages 1–6, 2018.

[13] Peter J. Denning, Douglas Comer, David Gries, Michael C. Mulder, Allen B. Tucker, A. Joe Turner, and Paul R. Young. Computing as a Discipline. *Commun. ACM*, 32(1):9–23, 1989.

[14] Vaibhav Bajpai, Anna Brunström, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis, Matthias Wählisch, and Klaus Wehrle. The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research. *Comput. Commun. Rev.*, 49(1):24–30, 2019.

[15] Alemnew Sheferaw Asrese, Magnus Boye, and Pasi Sarolahti. WePR: Web Performance and Rendering . `https://github.com/alemnew/wepr`, 2019.

[16] Alemnew Sheferaw Asrese. WebLAR: A Web Performance Measurement Tool. `https://github.com/alemnew/weblar`, 2019.

[17] Diego daHora, Alemnew Sheferaw Asrese, and Dario Rossi. Approximate ATF Chrome Extension. `https://github.com/TeamRossi/ATF-chrome-plugin`, 2018.

[18] Alemnew Sheferaw Asrese, Steffie Jacob Eravuchira, Vaibhav Bajpai, and Jörg Ott. Measuring Web Latency and Rendering Performance: Method, Tools & Longitudinal Dataset (Dataset). `https://github.com/alemnew/2019-tnsm-webperf-analysis`, 2019.

[19] Alemnew Sheferaw Asrese, Ermias Walelgne, Vaibhav Bajpai, Andra Lutu, Özgü Alay, and Jörg Ott. Measuring Web Quality of Experience in Cellular Networks (Dataset). `https://github.com/alemnew/2019-pam-weblar`, 2019.

[20] Nevil Brownlee and Kenneth C. Claffy. Internet Measurement. *IEEE Internet Computing*, 8(5):30–33, 2004.

[21] Brian Trammell, David Gugelmann, and Nevil Brownlee. Inline Data Integrity Signals for Passive Measurement. In *Traffic Monitoring and Analysis - 6th International Workshop, TMA 2014, London, UK, April 14, 2014. Proceedings*, pages 15–25, 2014.

[22] Stefano Traverso, Edion Tego, Eike Kowallik, Stefano Raffaglio, Andrea Fregosi, Marco Mellia, and Francesco Matera. Exploiting Hybrid Measurements for Network Troubleshooting. In *IEEE Networks*, Funchal, PT, September 2014. IEEE, IEEE.

[23] Carey L. Williamson. Internet Traffic Measurement. *IEEE Internet Computing*, 5(6):70–74, 2001.

[24] Michael Rabinovich and Mark Allman. Measuring the Internet. *IEEE Internet Computing*, 20(4):6–8, 2016.

[25] Jan Böttger. *Understanding Benefits of Different Vantage Points in Today's Internet*. PhD thesis, Technical University of Berlin, Germany, 2017.

[26] A. Morton. Active and Passive Metrics and Methods (with Hybrid Types In-Between). RFC 7799 (Informational), May 2016.

[27] Claude Chaudet, Eric Fleury, Isabelle Guérin Lassous, Hervé Rivano, and Marie-Emilie Voge. Optimal Positioning of Active and Passive Monitoring Devices. In *Proceedings of the 2005 ACM Conference on Emerging Network Experiment and Technology, CoNEXT 2005, Toulouse, France, October 24-27, 2005*, pages 71–82, 2005.

[28] Vaibhav Bajpai and Jürgen Schönwälder. A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts. *IEEE Communications Surveys and Tutorials*, 17(3):1313–1341, 2015.

[29] SamKnows. SamKnows. `https://samknows.com/`. Retrieved on Apr 18, 2019.

[30] Measurementlab. `https://measurementlab.net/`. Retrieved on Apr 18, 2019.

[31] Özgü Alay, Andra Lutu, Miguel Peón Quirós, Vincenzo Mancuso, Thomas Hirsch, Kristian Evensen, Audun Fosselie Hansen, Stefan Alfredsson, Jonas Karlsson, Anna Brunström, Ali Safari Khatouni, Marco Mellia, and Marco Ajmone Marsan. Experience: An Open Platform for Experimentation with Commercial Mobile Broadband Networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking, MobiCom 2017, Snowbird, UT, USA, October 16 - 20, 2017*, pages 70–78, 2017.

[32] Docker. `https://www.docker.com`. Retrieved on Apr 18, 2020.

[33] Speedcheker Ltd. SpeedChecker – Real World QoE Analytics for Telecoms and Regulators. `https://www.speedchecker.com/`. Retrieved on Mar 18, 2019.

[34] CAIDA. Archipelago (Ark) Measurement Infrastructure. `http://www.caida.org/projects/ark/`. Retrieved on Apr 18, 2019.

[35] RIPE Atlas Probes and Achors. `https://www.internetsociety.org/wp-content/uploads/2017/08/RIPE_Atlas_Brochure_Final_.pdf`. Retrieved on Jan 15, 2021.

[36] RIPE Atlas. `https://atlas.ripe.net/`. Retrieved on Jan 15, 2021.

[37] RIPE Atlas Project. `https://au.int/sites/default/files/documents/31361-doc-27-ripe-atlas-project.pdf`. Retrieved on Jan 15, 2021.

[38] Planetlab. `https://planet-lab.org/`. Retrieved on Apr 18, 2019.

[39] Sebastian Sonntag, Jukka Manner, and Lennart Schulte. Netradar - Measuring the Wireless World. In *11th International Symposium and Workshops on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, WiOpt 2013, Tsukuba Science City, Japan, May 13-17, 2013*, pages 29–34, 2013.

[40] OpenSignal. Meteor. `https://meteor.opensignal.com`. [Retrieved: May 12, 2019].

[41] PhoneLab. PhoneLab: A Smartphone Platform Testbed. https://www.phone-lab.org. [Retrieved: May 12, 2019].

[42] PacketLab. http://www.packet-lab.com. Retrieved on Mar 18, 2019.

[43] Dennis Fetterly, Mark S. Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of Web pages. *Softw., Pract. Exper.*, 34(2):213–237, 2004.

[44] Tom Callahan, Mark Allman, and Vern Paxson. A Longitudinal View of HTTP Traffic. In *Passive and Active Measurement, 11th International Conference, PAM 2010, Zurich, Switzerland, April 7-9, 2010. Proceedings*, pages 222–231, 2010.

[45] Sunghwan Ihm and Vivek S. Pai. Towards understanding modern web traffic. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011*, pages 295–312, 2011.

[46] Ben Newton, Kevin Jeffay, and Jay Aikat. The Continued Evolution of Web Traffic. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, USA, August 14-16, 2013*, pages 80–89, 2013.

[47] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016.

[48] HTTP Archieve. https://httparchive.org/. [Retrieved: Jun 22, 2020].

[49] Oliver Hohlfeld, Jan Rüth, Konrad Wolsing, and Torsten Zimmermann. Characterizing a Meta-CDN. In *Passive and Active Measurement - 19th International Conference, PAM 2018, Berlin, Germany, March 26-27, 2018, Proceedings*, pages 114–128, 2018.

[50] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. Web Content Cartography. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011*, pages 585–600, 2011.

[51] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the Performance of an Anycast CDN. In Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring, editors, *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 531–537. ACM, 2015.

[52] Stephen McQuistin, Sree Priyanka Uppu, and Marcel Flores. Taming Anycast in the Wild Internet. In *Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019*, pages 165–178. ACM, 2019.

[53] How Anycast Works - An Introduction to Networking. https://www.keycdn.com/support/anycast. [Retrieved: Jun 28, 2020].

[54] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. Flywheel: Google's Data Compression Proxy for the Mobile Web. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 367–380. USENIX Association, 2015.

[55] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), May 1996.

[56] A new image format for the Web. `https://developers.google.com/speed/webp/`. [Retrieved: Aug 29, 2019].

[57] Shailendra Singh, Harsha V. Madhyastha, Srikanth V. Krishnamurthy, and Ramesh Govindan. FlexiWeb: Network-Aware Compaction for Accelerating Mobile Web Transfers. In Serge Fdida, Giovanni Pau, Sneha Kumar Kasera, and Heather Zheng, editors, *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom 2015, Paris, France, September 7-11, 2015*, pages 604–616. ACM, 2015.

[58] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V. Madhyastha, and Vyas Sekar. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 439–453. USENIX Association, 2015.

[59] Evolution of HTTP. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP`. [Retrieved: Aug 18, 2019].

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), January 1997. Obsoleted by RFC 2616.

[61] A. Barth. HTTP State Management Mechanism. RFC 6265 (Proposed Standard), April 2011.

[62] J. Reschke. Hypertext Transfer Protocol (HTTP) Client-Initiated Content-Encoding. RFC 7694 (Proposed Standard), November 2015.

[63] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFCs 5785, 7230.

[64] R. Fielding (Ed.) and J. Reschke (Ed.). Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. RFC 7232 (Proposed Standard), June 2014.

[65] R. Fielding (Ed.), Y. Lafon (Ed.), and J. Reschke (Ed.). Hypertext Transfer Protocol (HTTP/1.1): Range Requests. RFC 7233 (Proposed Standard), June 2014.

[66] R. Fielding (Ed.), M. Nottingham (Ed.), and J. Reschke (Ed.). Hypertext Transfer Protocol (HTTP/1.1): Caching. RFC 7234 (Proposed Standard), June 2014.

[67] R. Fielding (Ed.) and J. Reschke (Ed.). Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235 (Proposed Standard), June 2014.

[68] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[69] M. Belshe, R. Peon, and M. Thomson (Ed.). Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard), May 2015.

[70] SMUX Protocol Specification. `https://www.w3.org/Protocols/MUX/WD-mux-980708.html`. [Retrieved: Aug 22, 2019].

[71] Ethan Blanton and Mark Allman. On making TCP more Robust to Packet Reordering. *Computer Communication Review*, 32(1):20–30, 2002.

[72] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency: the Virtue of Gentle Aggression. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 159–170. ACM, 2013.

[73] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing TCP's initial congestion window. *Computer Communication Review*, 40(3):26–33, 2010.

[74] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. TCP Fast Open. In *Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies, Co-NEXT '11, Tokyo, Japan, December 6-9, 2011*, page 21, 2011.

[75] Costin Raiciu, Christoph Paasch, Sébastien Barré, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 399–412, 2012.

[76] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.

[77] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A First Look at QUIC in the Wild. In *Passive and Active Measurement - 19th International Conference, PAM 2018, Berlin, Germany, March 26-27, 2018, Proceedings*, pages 255–268, 2018.

[78] Google. SPDY, an Experimental Protocol for a Faster Web. `https://www.chromium.org/spdy/spdy-whitepaper`. Retrieved on June 15, 2017.

[79] SPDY: An Experimental Protocol for a Faster Web. `https://www.chromium.org/spdy/spdy-whitepaper`. [Retrieved: Aug 24, 2019].

[80] R. Peon and H. Ruellan. HPACK: Header Compression for HTTP/2. RFC 7541 (Proposed Standard), May 2015.

[81] Torsten Zimmermann, Jan Rüth, Benedikt Wolters, and Oliver Hohlfeld. How HTTP/2 Pushes the web: An Empirical Study of HTTP/2 Server Push. In *IFIP Networking Conference, Stockholm, Sweden, June 12-16, 2017*, pages 1–9, 2017.

[82] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan R. Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 183–196, 2017.

[83] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC. In Phillipa Gill and Jana Iyengar, editors, *Proceedings of the Applied Networking Research Workshop, ANRW 2019, Montreal, Quebec, Canada, July 22, 2019.*, pages 1–7. ACM, 2019.

[84] Mike Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-22, IETF Secretariat, July 2019. `http://www.ietf.org/internet-drafts/draft-ietf-quic-http-22.txt`.

[85] QUIC Implementations. `https://github.com/quicwg/base-drafts/wiki/Implementations`. [Retrieved: Aug 18, 2019].

[86] Subodh Iyengar. Moving Fast at Scale: Experience Deploying IETF QUIC at Facebook. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, EPIQ'18, page Keynote, New York, NY, USA, 2018. Association for Computing Machinery.

[87] IETF-QUIC in IPv4. `https://quic.netray.io/stats.html`. [Retrieved: Aug 18, 2019].

[88] Amoghavarsha Suresh and Anshul Gandhi. Using Variability as a Guiding Principle to Reduce Latency in Web Applications via OS Profiling. In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo S. Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 1759–1770. ACM, 2019.

[89] Kit Eaton. How One Second Could Cost Amazon $1.6 Billion in Sales? `https://goo.gl/ZxLCa9`. Retrieved on Aug 16, 2019.

[90] Javad Nejati and Aruna Balasubramanian. An In-depth Study of Mobile Browser Performance. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 1305–1315, 2016.

[91] Jan Rüth and Konrad Wolsing and Klaus Wehrle and Oliver Hohlfeld. Perceiving QUIC: do users notice or even care? In Aziz Mohaisen and Zhi-Li Zhang, editors, *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, CoNEXT 2019, Orlando, FL, USA, December 09-12, 2019*, pages 144–150. ACM, 2019.

[92] Ermias Andargie Walelgne, Setälä Kim, Vaibhav Bajpai, Stefan Neumeier, Jukka Manner, and Jörg Ott. Factors Affecting Performance of Web Flows in Cellular Networks. In *17th International IFIP TC6 Networking Conference, Networking 2018, Zurich, Switzerland, May 14-16, 2018.*, pages 73–81, 2018.

[93] Yasir Zaki, Jay Chen, Thomas Pötsch, Talal Ahmad, and Lakshminarayanan Subramanian. Dissecting Web Latency in Ghana. In *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, pages 241–248, 2014.

[94] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. Analyzing the Costs (and Benefits) of DNS, DoT, and DoH for the Modern Web. In Phillipa Gill and Jana Iyengar, editors, *Proceedings of the Applied Networking Research Workshop, ANRW 2019, Montreal, Quebec, Canada, July 22, 2019.*, pages 20–22. ACM, 2019.

[95] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio M. Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The Cost of the "S" in HTTPS. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, pages 133–140, 2014.

[96] Jamshed Vesuna, Colin Scott, Michael Buettner, Michael Piatek, Arvind Krishnamurthy, and Scott Shenker. Caching Doesn't Improve Mobile Web Performance (Much). In *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016.*, pages 159–165, 2016.

[97] Sohaib Ahmad, Abdul Lateef Haamid, Zafar Ayyub Qazi, Zhenyu Zhou, Theophilus Benson, and Ihsan Ayyub Qazi. A View from the Other Side: Understanding Mobile Phone Characteristics in the Developing World. In *Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016, Santa Monica, CA, USA, November 14-16, 2016*, pages 319–325, 2016.

[98] Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian, Samir R. Das, and Michael Ferdman. Impact of Device Performance on Mobile Internet QoE. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*, pages 1–7, 2018.

[99] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. Characterizing Web Page Complexity and Its Impact. *IEEE/ACM Transaction on Networking*, 22(3):943–956, 2014.

[100] Addy Osmani. The Cost of JavaScript. `https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4`. [Retrieved: Aug 18, 2019].

[101] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying Page Load Performance with WProf. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, pages 473–485, 2013.

[102] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. Speeding up Web Page Loads with Shandian. In *13th USENIX Symposium on*

Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016, pages 109–122, 2016.

[103] Jordan Nielson, Carey Williamson, and Martin Arlitt. Benchmarking Modern Web Browsers. 04 2012.

[104] Jake Brutlag, Zoe Abrams, and Pat Meenan. Above the Fold Time: Measuring Web Page Performance Visually. *Velocity: Web Performance and Operations Conference*, mar 2011.

[105] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. HTTP/2 Prioritization and its Impact on Web Performance. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1755–1764, 2018.

[106] Utkarsh Goel, Moritz Steiner, Mike P. Wittie, Martin Flack, and Stephen Ludin. Measuring What is Not Ours: A Tale of 3^{rd} Party Performance. In Mohamed Ali Kâafar, Steve Uhlig, and Johanna Amann, editors, *Passive and Active Measurement - 18th International Conference, PAM 2017, Sydney, NSW, Australia, March 30-31, 2017, Proceedings*, volume 10176 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 2017.

[107] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 736–747. ACM, 2012.

[108] James Newman and Fabián E. Bustamante. The Value of First Impressions - The Impact of Ad-Blocking on Web QoE. In David R. Choffnes and Marinho P. Barcellos, editors, *Passive and Active Measurement - 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27-29, 2019, Proceedings*, volume 11419 of *Lecture Notes in Computer Science*, pages 273–285. Springer, 2019.

[109] Jia Wang. A survey of web caching schemes for the Internet. *Computer Communication Review*, 29(5):36–46, 1999.

[110] Wei-Guang Teng, Cheng-Yue Chang, and Ming-Syan Chen. Integrating Web Caching and Web Prefetching in Client-Side Proxies. *IEEE Trans. Parallel Distributed Systems*, 16(5):444–455, 2005.

[111] Byungjin Jun, Fabián E. Bustamante, Sung Yoon Whang, and Zachary S. Bischof. AMP up your Mobile Web Experience: Characterizingthe Impact of Google's Accelerated Mobile Project. In *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom'19, Los Cabos, Mexico, October 21-25, 2019*, 2019.

[112] Mohammad Ghasemisharif, Peter Snyder, Andrius Aucinas, and Benjamin Livshits. SpeedReader: Reader Mode Made Fast and Private. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 526–537, 2019.

[113] Sanae Rosen, Bo Han, Shuai Hao, Zhuoqing Morley Mao, and Feng Qian. Push or Request: An Investigation of HTTP/2 Server Push for Improving

Mobile Performance. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 459–468, 2017.

[114] Torsten Zimmermann, Benedikt Wolters, Oliver Hohlfeld, and Klaus Wehrle. Is the web ready for HTTP/2 server push? In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, pages 13–19, 2018.

[115] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K. Ramakrishnan. Towards a SPDY'ier Mobile Web? *IEEE/ACM Transactions on Networking*, 23(6):2010–2023, 2015.

[116] Lazy Loading Images and Video. `https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video/`. [Retrieved: Aug 28, 2019].

[117] Hernán Ceferino Vázquez, Alexandre Bergel, Santiago A. Vidal, Jorge Andrés Díaz Pace, and Claudia Marcos. Slimming JavaScript Applications: An Approach for Removing Unused Functions from JavaScript Libraries. *Information & Software Technology*, 107:18–29, 2019.

[118] Reduce JavaScript Payloads with Code Splitting. `https://developers.google.com/web/fundamentals/performance/optimizing-javascript/code-splitting/`. [Retrieved: Aug 28, 2019].

[119] Hyukwoo Park, Myungsu Cha, and Soo-Mook Moon. Concurrent JavaScript Parsing for Faster Loading of Web Apps. *Transactions on Architecture and Code Optimization*, 13(4):41:1–41:24, 2016.

[120] International Telecommunication Union. ITU-T Recommendation P10/G.100 : New Appendix I - Definition of Quality of Experience (QoE). Technical report, 2006.

[121] Patrick Le Callet, Sebastian Möller, and Andrew Perkis. Qualinet White Paper on Definitions of Quality of Experience (2012). Technical report, European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003), March 2013.

[122] Sebastian Egger, Tobias Hoßfeld, Raimund Schatz, and Markus Fiedler. Waiting Times in Quality of Experience for Web based Services. In *Fourth International Workshop on Quality of Multimedia Experience, QoMEX 2012, Melbourne, Australia, July 5-7, 2012*, pages 86–96, 2012.

[123] Jan-Niklas Voigt-Antons, Tobias Hoßfeld, Sebastian Egger-Lampl, Raimund Schatz, and Sebastian Möller. User Experience of Web Browsing - The Relationship of Usability and Quality of Experience. In *Tenth International Conference on Quality of Multimedia Experience, QoMEX 2018, Cagliari, Italy, May 29 - June 1, 2018*, pages 1–3, 2018.

[124] Nielsen, Jakob. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[125] Selim Ickin, Katarzyna Wac, Markus Fiedler, Lucjan Janowski, Jin-Hyuk Hong, and Anind K. Dey. Factors Influencing Quality of Experience of

Commonly used Mobile Applications. *IEEE Communications Magazine*, 50(4):48–56, 2012.

[126] Lamine Amour, Sami Souihi, Said Hoceini, and Abdelhamid Mellouk. A Hierarchical Classification Model of QoE Influence Factors. In *Wired/Wireless Internet Communications - 13th International Conference, WWIC 2015, Malaga, Spain, May 25-27, 2015, Revised Selected Papers*, pages 225–238, 2015.

[127] International Telecommunication Union. Recommendation ITU-T G.1031: QoE factors in web-browsing. Technical report, 2014.

[128] Tobias Hoßfeld, Sebastian Biedermann, Raimund Schatz, Alexander Platzer, Sebastian Egger, and Markus Fiedler. The Memory Effect and its Implications on Web QoE Modeling. 2011. Retrieved on Apr 10, 2020.

[129] Peter Reichl, Sebastian Egger, Sebastian Möller, Kalevi Kilkki, Markus Fiedler, Tobias Hoßfeld, Christos Tsiaras, and Alemnew Asrese. Towards a Comprehensive Framework for QOE and User Behavior Modelling. In *Seventh International Workshop on Quality of Multimedia Experience, QoMEX 2015, Pilos, Messinia, Greece, May 26-29, 2015*, pages 1–6, 2015.

[130] Tobias Hoßfeld, Sebastian Biedermann, Raimund Schatz, Alexander Platzer, Sebastian Egger, and Markus Fiedler. The Memory Effect and its Implications on Web QoE Modeling. In *23rd International Teletraffic Congress, ITC 2011, San Francisco, CA, USA, September 6-9, 2011*, pages 103–110, 2011.

[131] Sebastian Egger, Peter Reichl, Tobias Hoßfeld, and Raimund Schatz. "Time is Bandwidth"? Narrowing the Gap between Subjective Time Perception and Quality of Experience. In *Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10-15, 2012*, pages 1325–1330, 2012.

[132] Andreas Sackl, Pedro Casas, Raimund Schatz, Lucjan Janowski, and Ralf Irmer. Quantifying the Impact of Network Bandwidth Fluctuations and Outages on Web QoE. In *Seventh International Workshop on Quality of Multimedia Experience, QoMEX 2015, Pilos, Messinia, Greece, May 26-29, 2015*, pages 1–6, 2015.

[133] Qi Alfred Chen, Haokun Luo, Sanae Rosen, Zhuoqing Morley Mao, Karthik Iyer, Jie Hui, Kranthi Sontineni, and Kevin Lau. QoE Doctor: Diagnosing Mobile App QoE with Automated UI Control and Cross-layer Analysis. In *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, pages 151–164, 2014.

[134] Tobias Hoßfeld, Raimund Schatz, and Sebastian Egger. SOS: The MOS is Not Enough! In *Third International Workshop on Quality of Multimedia Experience, QoMEX 2011, Mechelen, Belgium, September 7-9, 2011*, pages 131–136, 2011.

[135] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad. Perceived Performance of Top Retail Webpages In the Wild. *Computer Communication Review*, 47(5):42–47, 2017.

[136] Alemnew Sheferaw Asrese, Steffie Jacob Eravuchira, Vaibhav Bajpai, Pasi Sarolahti, and Jörg Ott. Measuring Web Latency and Rendering Performance: Method, Tools, and Longitudinal Dataset. *IEEE Trans. Network and Service Management*, 16(2):535–549, 2019.

[137] Google Inc. Speed Index. `https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-Index`. Retrieved on Jun, 2019.

[138] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. Vesper: Measuring Time-to-Interactivity for Web Pages. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 217–231, 2018.

[139] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. Modeling Web Quality of Experience on Cellular Networks. In *The 20th Annual International Conference on Mobile Computing and Networking, MobiCom'14, Maui, HI, USA, September 7-11, 2014*, pages 213–224, 2014.

[140] Martino Trevisan, Idilio Drago, and Marco Mellia. PAIN: A Passive Web performance indicator for ISPs. *Computer Networks*, 149:115–126, 2019.

[141] Peter Reichl, Sebastian Egger, Raimund Schatz, and Alessandro D'Alconzo. The Logarithmic Nature of QoE and the Role of the Weber-Fechner Law in QoE Assessment. In *Proceedings of IEEE International Conference on Communications, ICC 2010, Cape Town, South Africa, 23-27 May 2010*, pages 1–5, 2010.

[142] Eva Ibarrola, Fidel Liberal, Ianire Taboada, and Rodrigo Ortega. Web QoE Evaluation in Multi-agent Networks: Validation of ITU-T G.1030. In *Fifth International Conference on Autonomic and Autonomous Systems, ICAS 2009, Valencia, Spain, 20-25 April 2009*, pages 289–294, 2009.

[143] W3C. W3C Web Performance Working Group. `https://www.w3c.org/webperf/`. Retrieved on Mar 18, 2019.

[144] Xiaoqian Wu. A Primer for Web Performance Timing APIs. `https://w3c.github.io/perf-timing-primer/`. Retrieved on Jul 21, 2017.

[145] Ilya Grigorik, Arvind Jain, and Jatinder Mann Mann. Page Visibility Level 2. `https://w3c.github.io/page-visibility`. Retrieved on Jul 15, 2017.

[146] Jatinder Mann and Jason Weber. Efficient Script Yielding. `http://w3c.github.io/setImmediate/`. Retrieved on Jul 15, 2017.

[147] WebPageTest – Web Performance and Optimization Test. `https://www.webpagetest.org/`. Retrieved on Mar 18, 2019.

[148] Sitespeed.io. Sitespeed. `https://www.sitespeed.io`. Retrieved on Aug 16, 2017.

[149] Marcel Duran. YSlow. `https://www.yslow.org`. Retrieved on Aug 6, 2017.

[150] Yahoo! Developers Network. Best Practices for Speeding Up Your Web Site. `https://developer.yahoo.com/performance/rules.html`. Retrieved on Aug 1, 2017.

[151] Pingdom. Pingdom Website Speed Test. https://tools.pingdom.com/. Retrieved on Aug 6, 2017.

[152] GT.net. GTMetrix: Website Spee and Performance Optimization. https://gtmetrix.com. Retrieved on Aug 16, 2017.

[153] Google Inc. PageSpeed Insights. https://developers.google.com/speed/pagespeed/insights/. Retrieved on Aug 16, 2017.

[154] Paul Irish. Pwmetrics. https://github.com/paulirish/pwmetrics. Retrieved on Aug 16, 2017.

[155] Google Inc. Lighthouse. https://developers.google.com/web/tools/lighthouse/. Retrieved on Aug 16, 2017.

[156] Artem Denysov. Easy Progressive Web Metrics. https://goo.gl/BhB69b. Retrieved on Aug 16, 2017.

[157] Cory Shaw, Nathan Probst, Jake Albaugh, Jamie Perkins, and Michael Neiling. PerfWars: Web Performance Battles. Compare Two URLs. Who Will Prevail? . https://perfwars.com. Retrieved on Aug 16, 2017.

[158] Avraam Mavridis. Louis. https://github.com/AvraamMavridis/gulp-louisa. Retrieved on Aug 1, 2017.

[159] Brain Jackson. Setting and Calculating a Web Performance Budget. https://www.keycdn.com/blog/web-performance-budget/. Retrieved on Aug 16, 2017.

[160] Tim Kadlec. What Does My Site Cost? https://whatdoesmysitecost.com. Retrieved on Jul 10, 2017.

[161] Uptrends: Free Website Speed Test. http://uptrends.com. Retrieved on Aug 1, 2017.

[162] Gaël Metais. Yellow Lab Tools. http://yellowlab.tools/. Retrieved on Jul 5, 2017.

[163] PhantomJS - Scriptable Headless Browser. http://phantomjs.org/. Retrieved on Jun, 2019.

[164] WebPerf IO. http://webperf.io. Retrieved on Aug 1, 2017.

[165] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the Edge Network. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, pages 246–259, 2010.

[166] Diana Joumblatt, Renata Teixeira, Jaideep Chandrashekar, and Nina Taft. HostView: Annotating end-host Performance Measurements with User Feedback. *SIGMETRICS Performance Evaluation Review*, 38(3):43–48, 2010.

[167] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. Measuring and Mitigating Web Performance Bottlenecks in Broadband Access Networks. In *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pages 213–226. ACM, 2013.

[168] Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson. Fathom: a browser-based network measurement platform. In *Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference, IMC '12, Boston, MA, USA, November 14-16, 2012*, pages 73–86, 2012.

[169] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, pages 417–429, 2015.

[170] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert G. Greenberg, and Yi-Min Wang. WebProphet: Automating Performance Prediction for Web Services. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*, pages 143–158, 2010.

[171] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R. Das. Improving User Perceived Page Load Times Using Gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 545–559, 2017.

[172] Vaspol Ruamviboonsuk Ravi Netravali, Muhammed Uluyol, and Harsha V. Madhyastha. Vroom: Accelerating the Mobile Web with Server-Aided Dependency Resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 390–403, 2017.

[173] Ravi Netravali and James Mickens. Prophecy: Accelerating Mobile Page Loads Using Final-state Write Logs. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 249–266, 2018.

[174] Apache. Apache JMeter. `https://jmeter.apache.org/`. Retrieved on Aug 6, 2017.

[175] Apache. Apache Bench. `https://httpd.apache.org/docs/2.4/programs/ab.html`. Retrieved on Aug 16, 2017.

[176] Httper. `https://github.com/httperf/httperf`. Retrieved on Aug 16, 2017.

[177] Heyman, Jonatan and Bystrom, Carl and Hamren, Joakim and Heyman, Hugo. LOCUST. `http://locust.io`. Retrieved on Aug 6, 2017.

[178] Joe Dog Software. Siege. `https://www.joedog.org/siege-home/`. Retrieved on Aug 10, 2017.

[179] Corey Goldberg. Multi-mechanize. `https://github.com/cgoldberg/multi-mechanize`. Retrieved on Aug 6, 2017.

[180] Theresa Enghardt, Thomas Zinner, and Anja Feldmann. Web Performance Pitfalls. In *Passive and Active Measurement - 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27-29, 2019, Proceedings*, pages 286–303, 2019.

[181] Alemnew Sheferaw Asrese, Ermias Andargie Walelgne, Vaibhav Bajpai, Andra Lutu, Özgü Alay, and Jörg Ott. Measuring Web Quality of Experience in Cellular Networks. In *Passive and Active Measurement - 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27-29, 2019, Proceedings*, pages 18–33, 2019.

[182] Resource Timing API. https://developer.mozilla.org/en-US/docs/Web/API/Resource_Timing_API. [Retrieved: Aug 28, 2019].

[183] Alemnew Sheferaw Asrese, Pasi Sarolahti, Magnus Boye, and Jörg Ott. WePR: A Tool for Automated Web Performance Measurement. In *2016 IEEE Globecom Workshops, Washington, DC, USA, December 4-8, 2016*, pages 1–6, 2016.

[184] Magnus Boye and Pasi Sarolahti. WebPerf: . https://github.com/wperf/webperf. Retrieved on Oct 25, 2017.

[185] HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer. . Retrieved on Aug 19, 2017.

[186] Diego da Hora, Alemnew Sheferaw Asrese, and Dario Rossi. Approximate ATF chrome extension. https://github.com/TeamRossi/ATF-chrome-plugin. [Retrieved: Aug 08, 2019].

[187] XVFB. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml. [Retrieved: May 12, 2019].

[188] Yi-Ching Chiu, Brandon Schlinker, Abhishek Balaji Radhakrishnan, Ethan Katz-Bassett, and Ramesh Govindan. Are We One Hop Away from a Better Internet? In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 523–529, 2015.

[189] Junaid Shaikh, Markus Fiedler, and Denis Collange. Quality of Experience from User and Network Perspectives. *Annales des Télécommunications*, 65(1-2):47–57, 2010.

[190] Demystifying Google Global Cache. https://blog.speedchecker.com/2015/11/30/demystifying-google-global-cache/. [Retrieved: Sep 17, 2019].

[191] Raul H. C. Lopes. *Kolmogorov-Smirnov Test*, pages 718–720. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[192] Anna Maria Mandalari, Andra Lutu, Ana Custura, Ali Safari Khatouni, Özgü Alay, Marcelo Bagnulo, Vaibhav Bajpai, Anna Brunström, Jörg Ott, Marco Mellia, and Gorry Fairhurst. Experience: Implications of Roaming in Europe. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, MobiCom 2018, New Delhi, India, October 29 - November 02, 2018*.

[193] Enrico Bocchi, Luca De Cicco, Marco Mellia, and Dario Rossi. The Web, the Users, and the MOS: Influence of HTTP/2 on User Experience. In *Passive and Active Measurement - 18th International Conference, PAM 2017, Sydney, NSW, Australia, March 30-31, 2017, Proceedings*, pages 47–59, 2017.

[194] Alexa – Top Sites. http://alexa.com/topsites. Retrieved on Jan 30, 2021.

[195] International Telecommunication Union. ITU-T Recommendation P.913: Methods for the subjective assessment of video quality, audio quality and audiovisual quality of Internet video and distribution quality television in any environment. Technical report, 1996.

[196] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alexander J. Smola, and Vladimir Vapnik. Support Vector Regression Machines. In *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 155–161, 1996.

[197] Wei-Yin Loh. Classification and Regression Trees. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 1(1):14–23, 2011.

[198] Yoav Freund and Robert E. Schapire. Experiments with a New Boosting Algorithm. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 148–156, 1996.

[199] Scikit-learn: Machine learning in python. `https://scikit-learn.org/stable/index.html`. [Retrieved: Nov 14, 2019].

[200] Masayuki Karasuyama and Ryohei Nakano. Optimizing SVR Hyperparameters via Fast Cross-Validation using AOSVR. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2007, Celebrating 20 years of neural networks, Orlando, Florida, USA, August 12-17, 2007*, pages 1186–1191. IEEE, 2007.

Errata

Publication II

- Page 544, Section VI.C: ‘an year-long’ is spelled incorrectly. It should be spelled as ‘a year-long’. The text should read as ‘We dissected the web latency distribution of a year-long subset of the dataset ... ’.

Publication IV

- Page 23, Section 2.3: ‘`imagemagic`’ is spelled incorrectly. It should be `imagemagick`. The text should read as ‘`imagemagick` [1] is used to compare the pixel difference...’.
- Page 25, Section 3.3: ‘NO_2 and NO_2’ is labelled incorrectly. One of them should be labeled as NO_3. The text should read as ‘*e.g.,* 0.46 between NO_2 and NO_3.

Aalto-DD 76/2021

ISBN 978-952-64-0400-4 (printed)
ISBN 978-952-64-0401-1 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Electrical Engineering
Department of Communications and Networking
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS