Web-based data driven network monitoring : from performance estimation to anomaly detection

Imane Taibi

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Imane TAIBI »

Web-based data driven network monitoring : from performance estimation to anomaly detection

Thèse présentée et soutenue à Rennes, le « 19 septembre 2022 »
Unité de recherche : Centre Inria Rennes-Bretagne Atlantique (Inria-Rennes)

Rapporteurs avant soutenance :

Pascal LORENZ	Professeur d'université de Haute Alsace
Abdelhamid MELLOUK	Professeur d'université Paris-Est Créteil

Composition du Jury :

Président :	Guillaume URVOY-KELLER	Professeur, Université Côte d'azur
Examinateurs :	Isabelle CHRISMENT	Professeure, Télécom Nancy
	Yassine HADJADJ-AOUL	Professeur, Université de Rennes 1
	Pascal LORENZ	Professeur, Université de Haute Alsace
	Abdelhamid MELLOUK	Professeur, Université Paris-Est Créteil
Dir. de thèse :	Gerardo RUBINO	Directeur de recherche, Inria Université de Rennes 1
Co-dir. de thèse :	Chadi BARAKAT	Directeur de recherche, Inria Côte d'Azur

Acknowledgement

First and foremost, I thank God for His blessings and for giving me the strength to undertake this work and complete it successfully. All praises to Allah.

I would like to express my deep and sincere gratitude to my supervisors, Dr. Chadi BARAKAT and Pr. Yassine HADJADJ-AOUL, for their dedicated support, guidance and encouragement. They were very patient with me during the moments of my struggles and helped me to overcome difficulties. Without their help, I wouldn't be able to reach today. I thank Dr. Gerardo RUBINO for his valuable advice and the jury for dedicating time to review my thesis.

I'm incredibly grateful to my parents, Abdellah TAIBI and Rekaya BOUKILI, for their love, prayers and caring. Thank you so much, mother and father. Thank you for your patience and for believing in me until the end. A warm thank you to my dear grandmother Fatima.

I would like to extend my heartfelt thanks to my beloved sisters, Hind and Fatima-Ezzahrae and my close friends for their support and encouragement throughout this thesis.

Finally, during this PhD, I have learnt many things, especially about myself, and have grown a lot. I realised that sometimes your most significant achievement is never to lose hope.

Abstract

Web browsing is one of the most widespread applications on the Internet that allows accessing a massive amount of services. Its accelerated and continuous growth increases the pressure on the Internet infrastructure and leads to unexpected and unwanted performance behaviours that negatively impact the quality of service delivered to the final customers. Therefore, network operators and service providers need to ensure that the quality of their services is guaranteed. It is also important for end-users to be informed about the reality of their network access, especially when the quality of their services of interest deteriorates. The solution to these challenges passes by proposing network monitoring solutions able to evaluate the performance of the underlying network, to detect anomalies and bottlenecks and to identify the root causes of performance degradation. Many network monitoring and troubleshooting tools have been recently proposed, in particular, the Web browser-based tools (e.g., Speedtest, Netalyzr, HMN, Fathom, etc.); these tools are able to monitor the Internet performance from the end-user point of view in a portable and easy way and to conduct representative end-to-end network measurements. However, these solutions are known to incur a high computational cost or exaggeratedly consume data. In fact, existing tools usually follow what is called the active measurement approach, which consists in injecting packet probes inside the network to achieve accurate and precise network measurements, such as latency and bandwidth. This additional measurement traffic is in one side non-negligible, limiting the frequency of usage of the tools, and on the other side it very likely disturbs the normal network conditions that the tools are trying to measure. The main purpose of this thesis is to leverage the passive measurements freely available in the browser and deep learning techniques to infer network performance without the addition of new measurement overhead. To that aim, we need to (i) understand the link between the Web browsing and the underlying Network, (ii) determine the Web metrics that influence the Web Quality of Experience (QoE), and (iii) propose solutions to infer network states from these Web metrics as precisely as possible.

We start by inferring the main properties of the underlying Network (in particu-

lar the delay, the bandwidth and the loss rate) from web performance metrics (e.g., Connect Start, Page Load Time, [11],[20]) using passive measurements obtained from within the browser. We use machine learning to calibrate algorithms that allow such inference. By comparing deep learning algorithms to classical Machine Learning (ML) algorithms like Random Forest, we highlight the feasibility of the task but also its complexity, hence the need for sophisticated deep learning algorithms such as convolutional neural networks (CNN). Then we study and examine the impact of Web complexity on estimating the two specific metrics, delay and downloading bandwidth. Moreover, and motivated by the concern of a fair comparison with existing web-based monitoring solutions, we propose an integrated framework where we implement our solution and mimic the behaviour of other solutions in a fully controlled environment. Later, we propose an original network monitoring framework based on Bayesian Gaussian Mixture Models (BGMM) coupled with an algorithm to detect in real-time the occurrence of network anomalies. All these contributions put together lead to an efficient, light-weight, and Web-based and data-driven network monitoring and troubleshooting solution that run as a plugin in the end users' browser.

Keywords: Network measurement, Network performance, Web browsing, Web performance, controlled experimentation, passive measurements, Quality of Service, prediction, Deep Learning, CNN, clustering, anomaly detection.

Résumé en Français

La navigation Web est l'une des applications les plus répandues sur Internet qui permet d'accéder à une quantité massive de services. Sa croissance accélérée et continue accentue la pression sur les infrastructures Internet et entraîne des performances inattendues et indésirables ayant un impact négatif sur la qualité des services fournis aux consommateurs. Par conséquent, les opérateurs de réseau et les fournisseurs de services doivent s'assurer que la qualité de leurs services est garantie. Il est également important que les utilisateurs soient informés de la réalité de leur accès au réseau, en particulier lorsque la qualité de leurs services se détériore. La solution à ces défis passe par la proposition de solutions de surveillance de réseau capables d'évaluer les performances du réseau sous-jacent, de détecter les anomalies et les blocages, et d'identifier les causes profondes de la dégradation des performances.

De nombreux outils de surveillance et de dépannage du réseau ont récemment été proposés, en particulier les outils basés sur le navigateur Web (e.g., Speedtest, Netalyzr, HMN, Fathom, etc.); ces outils sont capables de surveiller les performances d'Internet du point de vue de l'utilisateur final de manière facile et portable, et d'effectuer des mesures représentatives de l'accès réseau. Cependant, ces solutions sont connues pour leur coût de calcul élevé ou leur consommation exagérée de données. En fait, les outils existants suivent généralement ce qu'on appelle l'approche de mesure active, qui consiste à injecter des paquets à l'intérieur du réseau pour obtenir des mesures réseau précises, telles que la latence et la bande passante. Ce trafic de mesure supplémentaire est non négligeable, limitant la fréquence d'utilisation de type d'outils, et pouvant perturber les conditions de réseau que les outils tentent de mesurer.

L'objectif principal de cette thèse est de tirer parti des mesures passives librement disponibles dans le navigateur pour déduire les performances du réseau, le surveiller en temps réel et diagnostiquer ses dysfonctionnements sans ajouter de nouvelles surcharges. À cette fin, nous devons (i) comprendre le lien entre la navigation web et le réseau sous-jacent, (ii) déterminer les mesures web qui influent sur la qualité de l'expérience web (QoE), et (iii) proposer des solutions pour déduire l'état du réseau

à partir de ces mesures web le plus précisément possible.

Contributions de la thèse:

Les contributions les plus importantes de la thèse sont:

1. Nous proposons de déduire les principales propriétés du réseau sous-jacent à partir des mesures de performance web (e.g., Connect Start, Page Load Time [11], [20]) en se basant sur des mesures passives librement disponibles dans le navigateur. Nous utilisons l'apprentissage automatique (ML) pour calibrer les algorithmes qui permettent cette inférence aussi précisément que possible. En comparant les algorithmes de l'apprentissage profond aux algorithmes ML classiques comme Random Forest, nous soulignons la faisabilité de la tâche, mais aussi sa complexité, d'où le besoin d'algorithmes d'apprentissage profond sophistiqués comme les réseaux de neurones convolutifs (CNN). Cette approche nous a permis d'atteindre nos deux objectifs principaux : (i) réduire le coût des mesures actives à zéro, et (ii) déduire les caractéristiques du chemin vers le serveur web de notre choix sans recourir au déploiement d'un serveur de mesures dédié, comme c'est le cas aujourd'hui avec les outils existants.

   Nos modèles d'apprentissage profond sont calibrés à l'aide d'une approche d'expérimentation contrôlée où nous modifions artificiellement les conditions du réseau et automatisons une activité de navigation sur le web, puis nous utilisons la vérité terrain sur l'état du réseau avec les mesures passives disponibles dans le navigateur pour la formation et la validation de nos modèles. Nous préparons le terrain pour cette modélisation en effectuant une analyse de sensibilité pour comprendre la dépendance entre la performance au niveau du web et celle du réseau. Plus en détail, nous concevons une méthodologie pour collecter un grand ensemble de données qui relie les mesures de performance web aux conditions de réseau sous-jacent sur un exemple de trois mesures de réseau, à savoir le temps aller-retour (RTT), la bande passante de téléchargement et le taux de perte. Nous montrons que nous pouvons les estimer avec une bonne précision en utilisant uniquement des mesures passivement obtenues à partir du navigateur, en particulier avec le modèle CNN qui surpasse les techniques d'apprentissage automatique (ML) traditionnelles.
2. Compte tenu de l'influence du processus de chargement des pages web, nous

étudions et examinons cet impact sur nos modèles d'estimation, notamment, le délai et la bande passante de téléchargement. Pour ce faire, nous capturons des mesures spécifiques au web, avec de l'information sur les caractéristiques des pages Web visitées (par exemple, la taille de la page Web, le nombre d'objets, etc.) ainsi que les protocoles pris en charge (par ex. HTTP/1.1, HTTP/2, HTTPS, etc.). Ensuite, nous appliquons la modélisation du réseau neuronal convolutif (CNN) pour estimer l'état du réseau à partir de ces mesures passives. En outre, et aux fins de comparaison avec les solutions de surveillance en ligne existantes, nous proposons un framework intégré dans laquelle nous mettons en œuvre notre solution et reproduisons le comportement d'autres solutions pour assurer une comparaison équitable entre les différentes approches dans un environnement entièrement contrôlé. Nos résultats montrent que les caractéristiques des pages web influent sur l'estimation des mesures de rendement du réseau; par conséquent, elles peuvent nous permettre de connaître la façon de choisir les pages web pour en accroître l'exactitude. En outre, notre méthodologie est en mesure de fournir une très bonne estimation de la performance du réseau sous-jacent, surpassant parfois certaines solutions existantes, en particulier lorsque les paramètres du réseau sont de grandes valeurs.

3. Nous proposons une solution légère pour surveiller la performance du réseau basée sur la consolidation de notre solution introduite plus tôt afin de différencier entre les différentes conditions du réseau qui font face aux pages web visitées par l'utilisateur. Notre contribution consiste en (i) un framework original de surveillance du réseau, basée sur les modèles de mélange gaussiens bayésiens (BGMM), capable de fournir de l'information sur l'état sous-jacent du réseau pour les différentes pages Web visitées par l'utilisateur, et (ii) un algorithme permettant de détecter en temps réel l'apparition d'anomalies et d'identifier les pages web qui en sont affectées, conduisant ainsi à une solution de dépannage de navigation web efficace.

Organisation de la thèse:

Cette thèse est organisée comme suit.

1. Dans le chapitre 2, nous présentons l'état de l'art des différents aspects liés au web. Ensuite, nous revisitons le thème de la modélisation de la qualité

des réseaux à partir de métriques de performance web en mettant en avant les principales approches utilisées par les chercheurs, ainsi que les différentes métriques web et réseaux. Enfin, nous examinons la documentation sur la surveillance des réseaux et la détection des anomalies concernant les outils et les approches de mesure disponibles.

2. Dans le chapitre 3, nous décrivons en détail notre approche pour estimer les métriques réseau sous-jacent à partir des mesures de performance web ainsi que le processus de collecte de données. Pour cela, nous concevons une méthodologie pour collecter un grand ensemble de données qui relie les mesures de performance web aux mesures du réseau sous-jacent. Ensuite, nous présentons notre modèle de réseaux de neurones convolutifs (CNN) pour calibrer les algorithmes qui permettent l'inférence de l'état du réseau, aussi précisément que possible. Ensuite, nous validons notre solution et étudions son efficacité en la comparant avec deux techniques de ML connues : les réseaux neuronaux et les modèles de forêts aléatoires, et nous discutons les résultats de nos expériences.
3. Dans le chapitre 4, et compte tenu de l'influence de la complexité du web sur la performance du réseau, nous étudions et examinons cet impact sur l'estimation des performances du réseau. Nous proposons de capturer des mesures plus spécifiques que nous étendons avec des informations sur les caractéristiques des pages Web visitées ainsi que les protocoles pris en charge, et nous appliquons l'algorithme d'apprentissage profond à base de CNNs pour estimer l'état du réseau à partir de ces mesures passives. Ensuite, nous analysons l'impact de la complexité du web sur l'estimation. De plus, aux fins de comparaison avec les solutions de surveillance Web existantes, nous présentons notre framework intégré dans lequel nous mettons en œuvre notre solution et reproduisons le comportement d'autres solutions. Enfin, nous montrons comment notre méthodologie est capable de fournir une très bonne estimation de la performance du réseau sous-jacent, surpassant parfois certaines solutions existantes.
4. Dans le Chapitre 5, nous commençons par proposer un framework de surveillance de réseau original, basé sur les modèles de mélange gaussiens bayésiens (BGMM), capable de fournir des informations sur l'état de réseau sous-jacent pour les différentes pages Web visitées par l'utilisateur. Ensuite, nous validons notre solution et étudions son efficacité. Finalement, nous proposons un algorithme capable de surveiller le réseau et de détecter l'apparition d'anomalies

en temps réel, conduisant ainsi à une solution de dépannage de navigation web efficace.

5. Dans le chapitre 6, nous concluons la thèse, et nous finissons par illustrer quelques perspectives de travaux futurs.

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ATF	Above The Fold
BGMM	Bayesian Gaussian Mixture Models
BI	ByteIndexe
CDN	Content Delivery Network
CNN	Concolutional Neural Networks
CSS	Cascading Style Sheets
DL	Deep Learning
DNS	Domain Name System
DOM	Document Object Model
GMM	Gaussian Mixture Models
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
IP	Internet Protocol
ML	Machine Learning
MOS	Mean Opinion Score
NN	Neural Networks
PLT	Page Load Time
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RF	Random Forest
RTT	Round Trip Time
SSL	Secure Sockets Layer

TCP	Transfer Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WebQoE	Web Quality of Experience

Chapter 1

Introduction

1.1 General context

The internet has witnessed a massive development in a short period of time, until it has become today a global entity that extends to all different fields of life such as education, business, communication, and entertainment. The internet is the culmination of many efforts and contributions. Its history begins in 1969 with the creation of a large network that interconnects many computers. Then, Ray Tomlinson sent the first message through e-mail in 1971, and used "@" to distinguish between the name of the sender and the name of the network in the e-mail address. Two years later, the internet Transmission Control Protocol (TCP/IP) was designed and in 1983 the protocol became the standard for communication between computers.

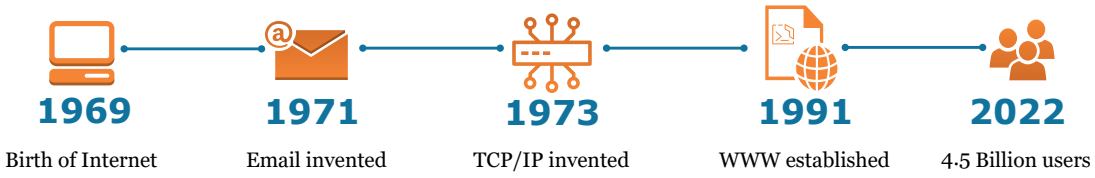


Figure 1.1 – Internet evolution timeline

Web browsing is one of the most dominant activities on the internet that allows accessing a massive number of services, from watching videos and news, searching for information, to online shopping, etc. In other words, the web has become over the years a principal pillar of the information age. However, the accelerated and continuous growth of web traffic can still lead to unexpected and unwanted performance behaviours that negatively impact the quality of service delivered to the customer, thus leading to session and even service abandonment. Indeed, recent studies have shown that end-user performance is directly related to the number of visitors and sales on popular web pages. Consistently, high web page delay increases the abandonment rate by end users. For example, according to Amazon, a 100ms additional

delay in page load time can cause a 1% decrease in sales [25], and for Google adding 500ms in latency can lead to a drop of 20% in their traffic. In addition, for a small-scale e-commerce web page with daily sales of $100,000, a 3-second page delay can lead to about 21% loss in sales annually [82]. These implications show the importance and the potential economic value of detecting web performance problems, and particularly those of the underlying network infrastructure.

Monitoring network performance in the context of web browsing activities becomes, therefore, a necessity today, especially during the COVID-19 pandemic, where internet traffic has increased significantly due to the sanitary lockdowns (see figure 1.2). On one hand, this monitoring allows network operators and content providers to evaluate and understand the quality they deliver to their customers, and gives them hints on the origin of any issues within the network. On the other hand, performing network measurements gives the end users a better understanding of the actual state of their networks and the services to which they connect, especially when the quality of service drops; this gained knowledge allows them to take the appropriate actions towards improving the quality of their communications [79], [49], [3], [30], [10].

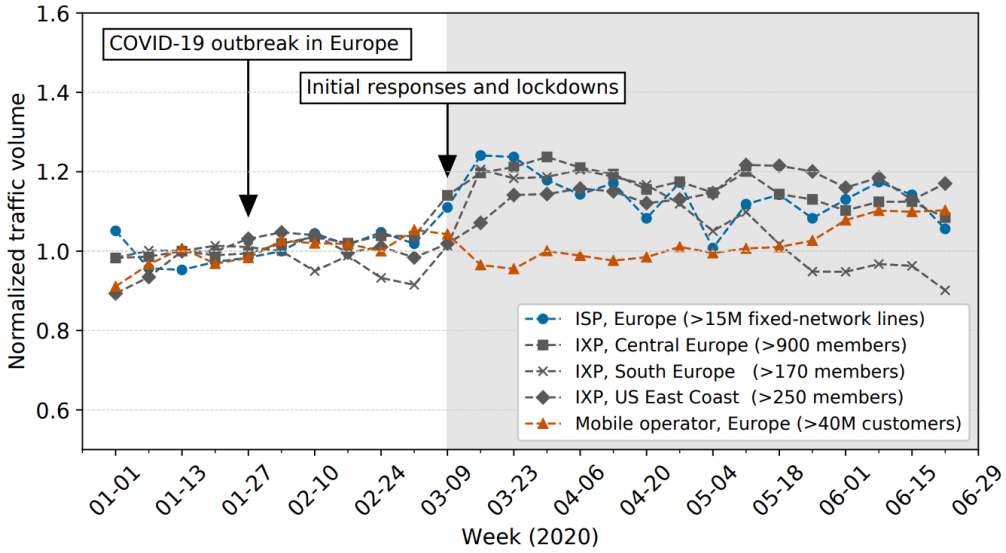


Figure 1.2 – Internet traffic changes during the outbreak of COVID19 at multiple vantage points according to Feldmann et al. [28]

1.2 Challenges and Motivations

1.2.1 Web-based network monitoring

Many monitoring platforms and tools have been proposed to monitor the status of the internet, some of which are standalone applications. As the final user is the one who best perceives the quality received, there has been in recent years an emerging need to monitor the internet performance from the end user point of view in a portable and easy way. This has led to a new trend of network measurement tools [57], [48], [23],[83],[53] that can run within the browser itself – we talk about web-based network monitoring, known by their portability and the facility with which they are used, and allowing to easily establish the link between web and network performance while staying within the web browser environment.

In general, bridging the gap between web performance and network performance requires the deployment of either standalone measurement tools in the form of applications (e.g., pathload for bandwidth measurement [47]), or of browser plugins with an explicit action by the user in the form of visiting a web page to download a file, for example [95]. For the latter, many web measurement tools have recently been proposed to conduct measurements from within the browser itself. For instance, we find Speedtest.net [83], a web site using Flash and Java Applet to measure the ping (latency), the download speed and the upload speed. Netalyzr [53] is a Java Applet that probes many servers and provides information on the network and its access services. Fathom [23] is a Firefox plugin that performs several passive and active network measurements using JavaScript.

These solutions are either computationally expensive, less generic, or greedy in terms of data consumption. In fact, existing tools usually implement an active approach. By injecting dedicated traffic into the network, they are able to derive accurate measurements with high precision about the network performance, such as the Round-Trip Time (RTT) and the download speed. These tools are, however, known to incur cost on the network and the data plane of the end user, especially in a mobile setup, with possible interaction of the measurement traffic with the traffic of the other applications and users, which if it happens impacts both the other traffic and the accuracy of the measurement results themselves.

The question of reducing this cost is then open, with some of the tools [5] relaying for example on lightweight UDP-based probing instead of TCP-based probing [23], to

reduce the cost of downloading or pushing MegaBytes of files to estimate the network throughput with TCP at the risk of being filtered or shaped by the network. Moreover, active measurement tools, especially the ones dedicated to bandwidth measurements, require the collaboration of servers on the other side from which to download files or to which to upload files. The deployment of these measurement servers is a difficult task, as theoretically they have to be present everywhere if we want to be general and accurate in our path performance measurements. The common practice is to deploy them close to the edge, which results in limiting the spectrum of active measurements to the path between the end-user device and the nearest server, thus implicitly covering the access network portion.

From these observations, and for web browsing in particular, it turns out that there is a need to deploy techniques able to carry out network measurements at low or even zero cost to the web server of our choice. Hence, the interest of resorting to native browser level passive measurements whenever possible, and to support them by ML-based algorithms to improve the accuracy of their estimation. This is our main idea in this thesis.

1.2.2 Web and Network performance relationship

Understanding the relationship between web performance and the actual network state is key to network troubleshooting. However, given the complexity of the web [14], this task is very challenging [79]. Indeed, today's web pages incorporate plenty of objects fetched from multiple servers through multiple connections and use complicated rendering technologies with advanced underlying protocols such as HTTP 1.1, HTTP/2 and QUIC (to add to transport and mac protocols). Choosing the web metrics that faithfully reflect the end-user Quality of Experience (QoE) is another tough task [11]. The PLT (Page Load Time) is one of the commonly used metrics. For example, Alexa reports the quantiles of the PLT, and Google uses PLT to rank search results. However, recent studies deduce that this metric alone cannot give a precise estimation of web browsing quality, hence the need for finding more suitable metrics that are closer to screen rendering and its subjective evaluation by the user [20]. That is why new metrics such as SpeedIndex and Above The Fold (ATF) render time have seen the light.

In this thesis, we will seek the most relevant web metrics while being as exhaustive as possible. Then, we propose a data-driven methodology based on controlled

experimentation and machine learning (ML) to establish consolidated links between what happens at the Web level and the performance conditions of the underlying network.

1.2.3 Network performance anomaly detection

Detecting performance anomalies and identifying potential root causes are challenging, mainly because of the scale, heterogeneity and dynamics of today's internet infrastructure. A timely detection of performance anomalies is needed to guarantee network and web performance, service reliability, and Quality of Service (QoS), and this is critical for service providers before anomalies trigger unexpected service downtime. Considerable efforts have been made to address this issue in the literature. Many of the proposed solutions focus on solving the problem in specific domains by leveraging the power of statistical and machine learning techniques [36],[21], [99], [44], [107], [100]. In fact, a basic network anomaly detection system monitors the performance behaviours of the underlying network and collects vital measurements to create baseline models or profiles of typical network behaviours. It continuously monitors new measurements for deviations in order to detect expected or unexpected performance anomalies and carries out root-cause analysis to identify associated bottlenecks.

Network performance anomaly detection is often carried out with network-level data; moving it to the web browser level and seeking solutions to detect anomalies with web measurements is a challenging task that we handle in this thesis.

1.3 Contributions of the Thesis

The main objective of this thesis is to leverage passive measurements freely available in the browser and deep learning techniques to infer network performance without the addition of new measurement overhead. For this purpose, several issues have been addressed. The most significant contributions of the thesis are:

1. We propose to infer the main properties of the underlying network from web performance metrics (e.g., Connect Start, Page Load Time, [11],[20]) based on passive measurements obtained from within the browser. We use machine learning to calibrate algorithms that allow such inference as accurately as pos-

sible. By comparing deep learning algorithms to classical ML algorithms as Random Forest, we highlight the feasibility of the task, but also its complexity, hence the need for sophisticated deep learning algorithms as convolutional neural networks (CNN). This approach allowed us to achieve our two main goals: (i) reducing the cost of active measurements to zero, and (ii) inferring the characteristics of the path to the web server of our choice without resorting to the deployment and choice of a dedicated measurement server, as it is the case nowadays with existing tools.

Our deep learning models are calibrated using a controlled experimentation approach where we artificially change the network conditions and automate a web browsing activity, then we use the ground truth on the network state together with the passive measurements available in the browser for the training and validation of our models. We prepare the ground for this modelling by carrying out a sensitivity analysis to understand the dependency between the performance at the web level and the one of the network.

In more detail, we engineer a methodology for collecting a large dataset that links the web performance measurements to the underlying network conditions over an example of three network metrics, namely the round-trip time (RTT), the download bandwidth and the loss rate. We show that we can estimate them with good accuracy by only using measurements passively obtained from within the browser, especially with the CNN model that outperforms the traditional Machine Learning (ML) techniques.

2. Considering the influence of the loading process of web pages, we study and examine this impact on our estimation models, in particular, delay and download bandwidth. To do so, we capture web-specific measurements that we extend with information on the characteristics of the visited web pages (e.g., web page size, number of objects, etc.) as well as the protocols supported (e.g., HTTP/1.1, HTTP/2, HTTPS, etc.). Next, we apply Convolutional Neural Network modelling (CNN) to estimate the network state from these passive measurements. Moreover, and for the purpose of comparison with existing web-based monitoring solutions, we propose an integrated framework where we implement our solution and mimic the behaviour of other solutions to ensure a fair comparison between the different approaches in a fully controlled environment. Our results show that web page characteristics impact the estimation of

network performance metrics; thus, it can give us hints on how to choose web pages to increase accuracy. Furthermore, our methodology is able to provide a very good estimation of the underlying network performance, outperforming some existing solutions by times, especially when the network metrics are of large values.

3. We propose a lightweight solution for monitoring network performance based on consolidating our solution introduced earlier and building upon it towards differentiating between the different network conditions that face the web pages visited by the user, which is essential for network anomaly detection and web browsing troubleshooting. Our contribution consists of (i) an original network monitoring framework, based on Bayesian Gaussian Mixture Models (BGMM), able to provide information on the underlying network state for the different visited web pages by the user, and (ii) an algorithm to detect in real time the occurrence of anomalies and identifies the web pages that are affected by them, thus leading to an efficient web browsing troubleshooting solution.

1.4 Thesis roadmap

The rest of the thesis is organized as follows.

1. In Chapter 2, we overview the state of the art of the different aspects related to the web, from web basics and evolution to web performance. Then, we revisit the topic of modelling network quality based on web performance metrics by highlighting the main approaches used by researchers, as well as the different web and network metrics. Finally, we review the literature on network monitoring and anomaly detection regarding the available measurement tools and approaches.
2. In Chapter 3, we describe in detail our approach for estimating the underlying network metrics from web performance measurements, as well as the data collection process. For that, we engineer a methodology for collecting a large dataset that links web performance measurements to underlying network measurements. Next, we present our convolutional neural networks (CNN) model to calibrate algorithms that allow network inference as accurately as possible. Then, we validate our solution and study its efficiency by comparing it with

two known ML techniques: Neural Networks and Random Forests models, and we discuss the results of our experiments.

3. In Chapter 4, and given the influence of web complexity on web and network performance, we study and examine this impact on the estimation of network performance, in particular, delay and download bandwidth. We propose to capture more specific measurements that we extend with information on the characteristics of the visited web pages as well as the protocols supported, and we apply the CNN deep learner to estimate the network state from these passive measurements. Then, We analyze the impact of web complexity on the estimation. Also, for the purpose of comparison with existing web-based monitoring solutions, we propose an integrated framework where we implement our solution and mimic the behaviour of other solutions to ensure a fair comparison between the different approaches in a fully controlled environment. Finally, we show how our methodology is able to provide a very good estimation of the underlying network performance, outperforming some existing solutions by times.
4. In Chapter 5, we start by proposing an original network monitoring framework, based on Bayesian Gaussian Mixture Models (BGMM), able to provide information on the underlying network state for the different visited web pages by the user. Next, we validate our framework and study its efficiency. Then, we suggest an algorithm able to monitor the network and detect the occurrence of anomalies in real-time, thus leading to an efficient web browsing troubleshooting solution.
5. In Chapter 6, we conclude the thesis, and we present perspectives and future directions for our work.

Chapter 2

State of the art

In this chapter, we first provide an overview of the web as well as the basics of its structure, then we highlight the main contributions in the literature related to network performance monitoring and troubleshooting tools. Finally, we discuss the different aspects related to inferring network states from web performance metrics.

2.1 An overview of the web

The internet and the web have affected our lives greatly by changing radically the way we seek, exchange, distribute, and process information. In few decades, the internet has become a powerful social force, impacting all areas of life, constituting the most extensive library and largest marketplace on earth.

Although common usage tends to confuse the words "internet" and web, we should distinguish between them. Indeed, the internet is a global communication network linking billions of computers, which offers a vast range of services. It is a general term relative to the underlying networks that compose the internet and the way they are connected together. On the other hand, the web is related to the information stored and available on the internet: a huge distributed and dynamic system composed of many web pages.

2.1.1 Web basics

The WWW or the World Wide Web [8] is one of the most dominant services running on the internet. We can define it as a set of web pages spread over many sites. These pages, which are constituted of text, images, sound, etc., are linked together via hypertext links, hence the possibility of surfing from one page to another. Accessing information of a web site is managed by a web server. Users retrieve the content of a website using a web client, which is also called a browser such as Mozilla Firefox or Google Chrome. Accordingly, the web is implemented based on three elements: resources, resource identifiers, and transfer protocols that depict respectively how information is stored, located, and exchanged.

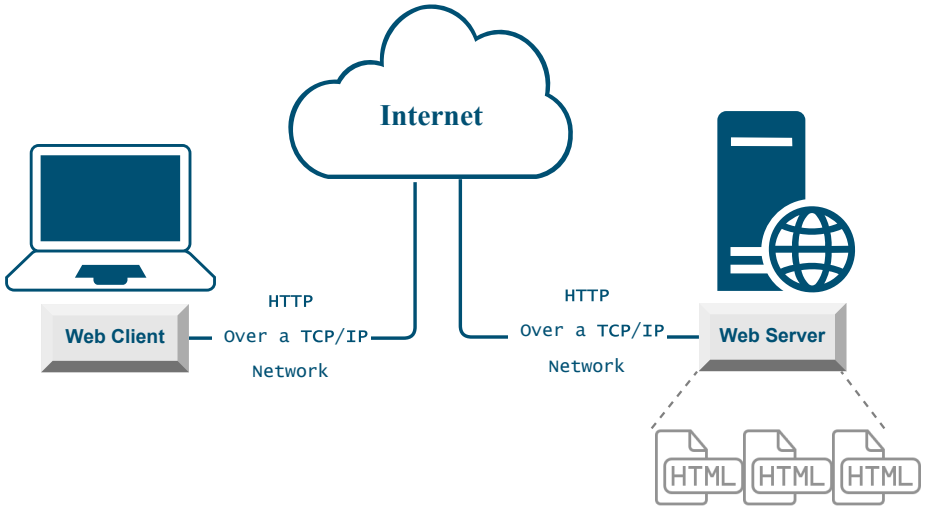


Figure 2.1 – Basic Web concepts

Web resources. A web resource is a basic building block of the World Wide Web architecture. Resources are entities such as text, images, audio, etc., accessible independently from each other. A vast proportion of existing web resources are HTML documents.

HTML, which stands for "HyperText Markup Language" [42], is a language used to encode a web document's content and include hyperlinks to create web pages.

HTML allows a website creator to manage how their web pages will display on a screen through the browser. It is based on a system of tags, making it possible to :

— Publish documents online with text, tables, lists, photos, etc.

— Bring information online via hypertext links with a simple click.

— Design forms to conduct transactions with remote services in order to search for information, make reservations, take orders, etc.

— Include spreadsheets, video or sound clips and other applications directly in documents.

Resource identifiers. Resource identifiers, or URLs (Uniform Resource Locator) [45], are commonly known as web addresses. A URL is a uniform string of characters that specifies the location of a web resource by indicating the name of the protocol used to retrieve it, the domain name that identifies a specific computer on the internet, and the access path which (i.e, a hierarchical description) indicates the location of the file in this computer.

Transfer Protocols. Transfer Protocols are conventions that regulate the exchange of information between two connected entities; a web client and a web server. The Hypertext Transfer Protocol, generally abbreviated as HTTP, is a client-server communication protocol developed by Tim Berners-Lee at CERN [17] as the basis for creating the World Wide Web. While HTML (Hypertext Markup Language) defines how a website is constructed, HTTP determines how the page is transmitted from server to client. Today, this protocol is used in a wide variety of ways:

— HTTP allows the browser to request all types of media used on modern websites: text, images, videos, source code, etc.

— Applications also use HTTP to load files and updates from remote servers.

— HTTP is also involved in REST APIs, a solution for controlling web services.

— In machine-to-machine communication, HTTP is used to communicate between web services.

— HTTP is also used by media players.

2.1.2 Web evolution

Figure 2.2 – Web evolution timeline

The process of consulting the information available on the internet with the hypertext protocol was imagined in 1989, at CERN in Switzerland [17]. This center can thus be considered as the creator of the Web. By creating the WorldWide Web software, Tim Berners-Lee [91] created both the first web browser and the first web editor because he wanted to make the web a collaborative medium in which all actors consult and create information. However, the web immediately turned into a medium for global information dissemination rather than simple collaboration.

In the first half of the 1990s, the concept of a website at the root of a stable domain name was not established. Sites were often set up in technical departments by employees and students, and URLs changed as people and infrastructure changed. Also, there was no effective search engine, and many pages were lists of links to the page author's favourite pages. In January 1994, Yahoo! was created and quickly became the largest web directory. In the second half of the 1990s, the web became popular, and all major companies, organizations, schools, administrations opened a website. Search engines became efficient, notably with the appearance of Altavista in December 1995, and finally Google in 1998. In this phase of media development, a flow of top-down information predominated: a website was made to disseminate the

information of its owner.

In the 2000s, the notions of blog, wiki and social networking (Myspace in 2003, Facebook in 2004) became popular. User-generated content spreads (Wikipedia in 2001, YouTube in 2005, Twitter in 2006). The Ajax technology (1998, theorized in 2005) is widely used to create complete applications that fit into a single web page (Google Maps in 2004). The expression Web 2.0, widely popularized in the mid-2000s, refers to this transition in the flow of information and the way of using the Web. The popularity of the term "Web 2.0" has led many people to call "Web 2.5", 3.0, 4.0, etc., their vision of the Web of the future.

2.1.2.1 Web content evolution

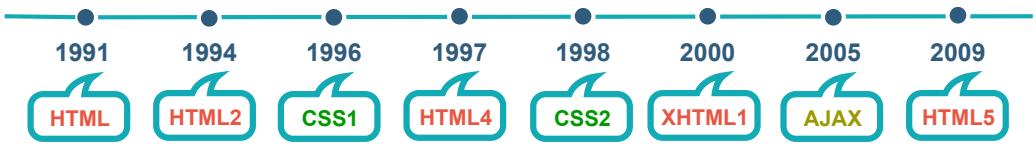


Figure 2.3 – Timeline of web technologies

The content of the web has changed drastically from "read-only mode" in 1990 [69] to a dynamic and interactive web, including media content of all sorts: images, audio, tables, etc. In order to handle these new requirements, many technologies have seen the light, mainly HTML2 alongside JavaScript in 1995, CSS1 (Cascaded Style Sheets) and XML (eXtensible Markup Language) in 1996, XHTML1 in 2000, AJAX (Asynchronous JavaScript and XML) in 2005 and HTML5 in 2009. Therefore, web content has kept evolving throughout the years. Several researchers have studied this evolution from different aspects: web page size in [12, 8], web page number of objects in [38, 29], and HTTP request/response size in [38, 16, 68]. All these studies agreed that, over the years, the number, and size of web pages had increased and become more and more complex.

2.1.2.2 Transfer protocols evolution

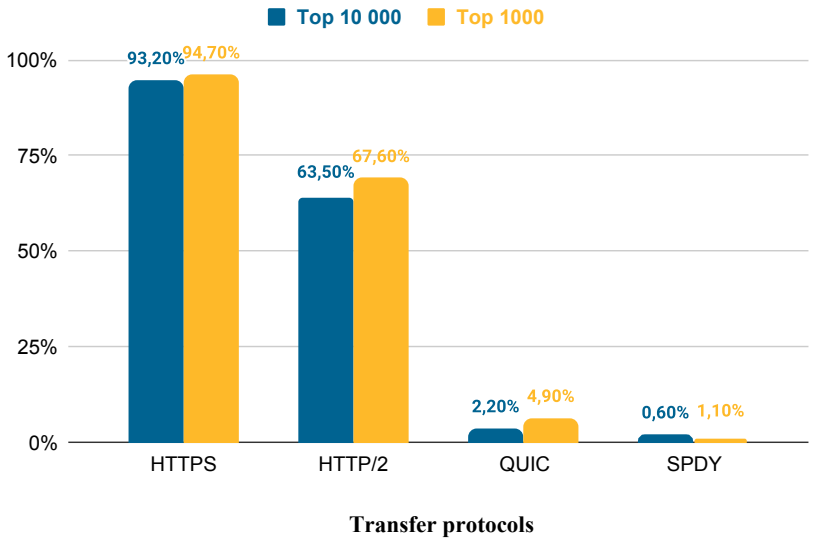


Figure 2.4 – Percentages of websites using HTTPs, HTTP/2, QUIC, and SPDY today according to W3Techs.com

The history of HTTP begins in 1989 [43]. The initial version of HTTP had version number 0.9 and was titled "one-line protocol". It only allowed viewing an HTML file from a server by simply transmitting the corresponding file. That's why this version of the protocol could only handle HTML files.

In 1996, the HTTP/1 version was described by the Internet Engineering Task Force (IETF) [7], but it was then only a non-binding proposal. A header was added to specify both the client request and the server response. In particular, the "Content-Type" header field, which allows files other than HTML documents to be transmitted, has been introduced.

In 1997, the HTTP/1.1 [31] version appeared, considered the first "official" standard and is still used today. Some of the most important new features compared to HTTP/1 are:

— Keep-alive: the client has the option of maintaining the connection beyond a request (persistent connection) by sending a keep-alive in the header of its

request.

— HTTP-Pipelining allows the client to send the next request before receiving the response to the first request.

— Cache: New mechanisms for buffering content are available.

Over the years, websites have steadily grown in size and complexity. To be able to load a modern website in a browser, this latter must request several megabytes of data and send up to 100 different HTTP requests. Since HTTP/1.1 allows for processing requests one after the other in the same connection, the more complex a website is, the more time it takes to establish the page. That's why Google has developed a new experimental protocol called SPDY [71](pronounced "Speedy"). The latter aroused great interest and resulted in the publication of the HTTP/2 version of the protocol in 2015 [6]. This new standard brings in particular new features, all aimed at speeding up the loading of websites. HTTP/2 is quickly becoming the norm; particularly, internet sites with high traffic did not wait to switch to this new version. In January 2020, nearly 42 percent of websites were using the HTTP/2 version, according to data from W3Techs.

In all older versions of the HTTP protocol, the underlying TCP transport protocol was one of the weak points. The latter requires the recipient of each data packet to confirm its reception, before the sender can send new packets (within the capacity of the congestion and receiver windows). But if one packet is lost, all other packets have to wait for the lost packet to be forwarded again. In this case, experts speak of "head-of-line blocking". Therefore, the new HTTP/3 must no longer be based on TCP but on UDP, which does not require any corrective measures of this type. The QUIC protocol (Quick UDP Internet Connections) [55] was developed for this purpose by relying on the UDP protocol and is expected to serve as the basis for HTTP/3 [9]. For the moment, HTTP/3 has not yet been definitively adopted by the IETF. But according to W3Techs, nearly 3 percent of websites use either QUIC or HTTP/3.

2.1.3 Web page loading process

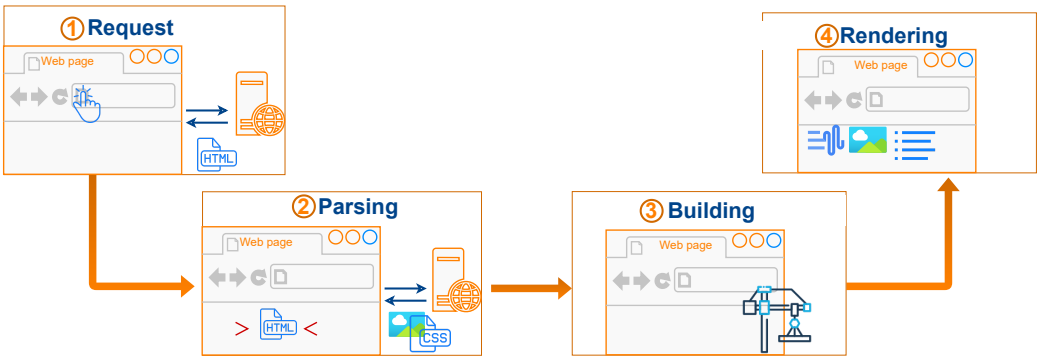


Figure 2.5 – How a web page is loaded and displayed

In this section, we provide an overview of what actually happens when a webpage is displayed in a browser. The process of loading and displaying a web page is based on four main steps. First, the web browser makes a request when the page link is clicked. Second, the web page and its resources are downloaded. Third, the web browser uses the fetched objects to create the page. And finally, the page is rendered to the end-user on the screen. In a nutshell, we refer to these steps as (i) request/response, (ii) parsing, (iii) building, and (iv) rendering (see Figure 2.5).

The request made for a web page usually occurs when a link is clicked, when the user types a URL in the Web browser, or when a page is refreshed. We call the moment the page is requested "Navigation Start" to indicate when the whole process of downloading and displaying a page begins. This step consists of fetching the main source of the web page, which is the HTML document located on the Web server using the HTTP transfer protocol. Then the Server provides the requested document. In the case of simple web pages, the loading will be complete by receiving the HTML file. However, most web pages need additional resources such as images, CSS, and JavaScript.

The purpose of the parsing step is to know whether the page needs additional resources. In particular, the Web browser will read or "parse" the HTML document

obtained and looks for any CSS, JavaScript and images to fetch. Then the browser makes another request to get each resource referenced by the page from the server (or other servers in case of external resources).

The building step consists of using the resources to create or "build" the web page. The web browser combines the information found in the HTML file and resources, and starts by building the "Document Object Model" or DOM. The DOM represents where objects are displayed on the page based on the HTML. Then, the browser builds the "CSS Object Map" or CSSOM, a map of the styles referenced by the CSS. Finally, it creates the Render Tree, which basically combines the DOM and the CSSOM to create a map of how the web page will be rendered.

Rendering is the final step at which the web browser will eventually display something on the screen. Here, the browser will start to determine the size of displaying each object; we talk about layout or reflow. Then, the browser will actually paint the whole page on the screen by converting each node of the render tree to pixels.

2.1.4 Web performance

In this section, we provide a brief overview of web performance along with the important factors affecting it. Web performance refers to several factors contributing to how quickly, efficiently, and correctly a web page loads. In other words, it can be defined by how fast a website is rendered on the screen of the user, which is becoming crucial nowadays. Indeed, according to Google, adding 500 *ms* in latency could lead to a drop of 20% in traffic [82]. Therefore, Google does not hide the fact that the speed of websites will have a direct impact on search rankings. Also, Amazon confirmed that a 100 *ms* additional delay in page load time could cause a 1% decrease in sales [25]. However, today's web pages incorporate plenty of objects fetched from multiple servers through multiple connections and use complicated rendering technologies with advanced underlying protocols such as HTTP 1.1, HTTP/2 and QUIC. This complexity increases the latency of loading web pages and consequently

degrades web performance, hence putting more pressure on servers, devices, and more importantly the underlying network infrastructure.

Next, we review the factors affecting the web performance, such as the web page complexity, the transfer protocols and the web content distribution infrastructure.

Webpage complexity. In Section 2.1.2.1, we explained how the web content has evolved, leading to having more complex web pages consisting of hundreds of web elements such as HTML, JavaScript, CSS, images, and multimedia content such as video and Flash. Several works have studied the impact of web complexity on web performance in terms of the number, the type, and the size of the web objects, as well as the processing of the objects and the rendering to the browser. For instance, Wang et al. [101] performed a study to capture the constraints between page parsing, JavaScript/CSS evaluation, and rendering activity. They find that inter-dependencies between HTML elements have a significant impact on the page load time and cannot be ignored, and that synchronous JavaScript plays a significant role in page load time by blocking HTML parsing. The authors, in [103], also showed that parsing-blocking CSS and JavaScript files slowed down the web page load time by 20%. Zhichun Li et al. [58] studied the impact of web object dependencies on predicting performance in an accurate and scalable manner. Butkiewicz et al. [15] demonstrated that the number and size of objects composing a web page can affect the download performance of the website. **Transfer protocols.** As we mentioned before, HTTP has evolved [43] from the first HTTP/0.9 to the HTTP/1.0 and sooner to the 'official protocol' HTTP/1.1 [31] with many important extensions namely the HTTP over TLS [74] (the secured version of HTTP). Then a new protocol SPDY [71] has been proposed by Google in order to tackle the inefficiencies of HTTP/1.1. Consequently, HTTP/2 has seen the light and standardized in 2015. Since this latter does not completely alleviate all the inefficiencies especially related to the TCP transport protocol, Google also proposed "Quick UDP Internet Connections" (QUIC) [55], a transport protocol that aims at reducing the network latency.

Many studies have been performed to understand performance-related inefficiencies in delivering web pages caused by transfer protocols. For instance, Liu et al. [59] studied the performance of HTTP/2 and HTTPS to show they could either decrease or increase the latency of loading web pages under several network conditions and page characteristics. Naylor et al. [65] also studied the impact of using HTTPS on latency. Regarding the QUIC protocol, Wolsing et al. [104] show that QUIC outperforms HTTP/2 over TCP in a comparison study. In addition, Rüth et al. [77] studied the impact of using QUIC protocol on end-users. Indeed, they perceive QUIC as a fast protocol compared to HTTP/2, especially in networks with a high delay and loss rate.

Web content infrastructure and caching. As stated previously, in order to ensure rapid availability on a global or near-global scale, Content Delivery Network (CDN) is needed; it consists of a network of servers that cache and delivers web data. Basically, CDN stores a cached version of websites around the world. In general, caching is the temporary storage of data to respond more quickly to future requests, and it can be used at several levels [97, 90]. Thus, the lack of a web content infrastructure and caching could negatively impact web performance. Indeed, Zaki et al. [108] found that the main causes for poor web performance in developing regions are related to the lack of good caching infrastructure. Also, Fanou et al. [27] showed that degraded web performance in Africa is related mainly to the inter-AS delays and the nonavailability of web content infrastructure. Regarding the importance of caching, Vesuna et al. [96] demonstrate that caching can improve the latency by 34% on desktops.

2.2 Network performance monitoring

Monitoring network performance is becoming crucial to detect anomalies and bottlenecks in the underlying network and identify the root causes of performance

degradation. Indeed, content providers need to ensure a good quality of their services. As for end-users, it is essential for them to know about the actual state of their networks and the services to which they connect, especially when the quality of service drops. Such knowledge allows them to take the appropriate actions toward improving the quality of their communications.

Many approaches have been proposed recently to measure the network performance and detect related performance issues. In this section, we present an overview of some troubleshooting techniques and tools used to monitor the network.

2.2.1 Network measurement approaches

Active measurements. This technique consists of injecting probing packets into the network in order to measure its characteristics, in particular the round trip time and the available bandwidth. Some of the traditional active network tools are: ping, traceroute and tcpdump. Unfortunately, this approach has many limitations:

1. When injected packets are lost, the tools become inefficient and the measurements inaccurate.
2. This method may introduce additional overhead to the network that may disturb the normal traffic flow.
3. The measurement results could be altered due to the extra traffic injected by the measurement tool itself, or by the other traffic flows.

Passive measurements. This technique consists of recording the internet traffic and studying its properties without injecting additional overhead, by simply gathering data passively. Unlike active measurements, passive methods do not introduce additional traffic; therefore, they do not disturb the flow or add extra overhead. Also, they provide a precise representation of the network traffic.

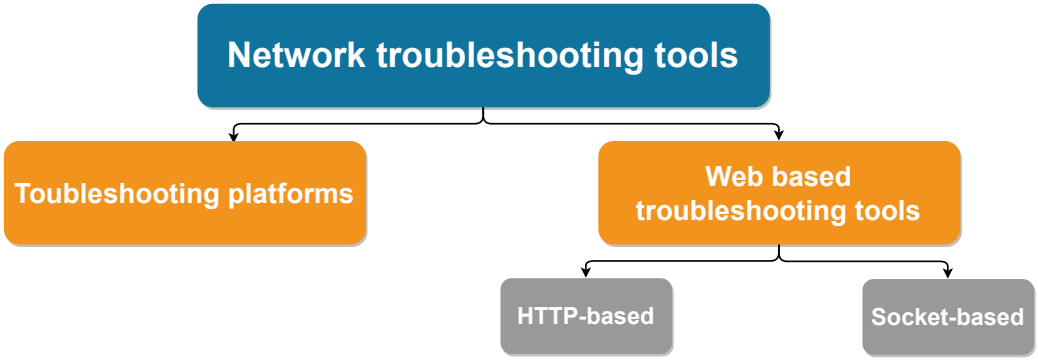


Figure 2.6 – Troubleshooting tools

2.2.2 Troubleshooting platforms and tools

Several measurement tools and approaches have been proposed for monitoring and troubleshooting the performance of the web ecosystem in the last few years. We can divide them into two main parts: network measurement platforms and web-based measurement tools [2][92], see Figure 2.6.

2.2.2.1 Troubleshooting platforms

A network measurement platform is a server-based infrastructure dedicated to test and to measure Internet and network performance using active or passive methods. Many researchers use PlanetLab [70] to support the development and testing of new network services, even if the latter is not really meant to be a measurement platform. Measurement Lab (M-Lab) [61], launched by Google, also allows and facilitates this kind of task. These platforms are a distributed set of servers hosted at research institutes.

The ACQUA application developed by the DIANA team at Inria Sophia Antipolis is another example of monitoring platforms based on crowd-sourcing [5]. ACQUA aims to estimate the quality of experience related to the different internet services, such as Skype and video streaming applications, from network-level measurements such as bandwidth, delay, and loss rate.

These standalone tools are known to be efficient. However, they require a tough

installation. Therefore, the necessity to easily perform measurements in end systems led to the development of new solutions: web-based measurement tools.

2.2.2.2 Web based troubleshooting tools

Many of these tools have recently seen the light mainly due to their ease of use and their portability and cross-platform features. For example, the Janc's method [48] provides multiple services based on Flash and JavaScript to measure the RTT and the throughput in a fully controlled way. Netalyzr [53], which is based on a Java applet accessible from a web page, provides measurements of the latency, the bandwidth and the buffering at the edge of the network. Speedtest [83], which operates mainly over TCP testing with an HTTP fallback for maximum compatibility, measures the latency, the download speed, and the upload speed. How's My Network (HMN) [76] is a website that provides predictions for common Internet activities. Table 2.1 gives a summary of the main tools and the approach they follow. In this thesis, we position ourselves on the front of web measurements tools, given their increased level of practicality and interest for the end user.

Table 2.1 – Main network troubleshooting tools and their approaches

Troubleshooting tool	Nature	Methodology	Approach
Speedtest.net	Website, plugin	Flash	HTTP
Janc's study	Native	JavaScript	HTTP
Netalyzr	plugin	Java Applet	Socket
HMN	plugin	Java Applet	Socket
NDT	plugin	Java Applet	Socket
Fathom	plugin	Java Script	HTTP
Navigation timing API	Native	DOM	HTTP

In general, the methods to measure the network from within the browser (e.g., delay, bandwidth) consist of recording the time before and after retrieving an object from the web server. The measurement process used by these methods can therefore

be HTTP-based or socket-based.

HTTP-based. HTTP-based methods are implemented through JavaScript or Flash. One way of doing this is by using the JavaScript function `Date.getTime()` in the date API, with DOM [24] to fetch the wanted resource. This technique is very simple and supported by all the browsers. Another method is to use the XHR (`XMLHttpRequest`) API [105] to request objects using JavaScript's browser environment. Alternatively, one can also use Flash, in particular the `URLLoader` class, which is able to download data from a URL as text, binary data, or URL-encoded variables. It is useful for downloading text files, XML, or other information to be used in a dynamic data-driven application.

Socket-based. On the other hand side, socket-based methods are implemented through TCP or UDP connections to exchange binary data. Flash Action Script for example, can create TCP sockets, using the following APIs: `Socket`, `SecureSocket`, `ServerSocket` as well as `XMLSocket`. One other way is the use of Websockets, which are based essentially on HTTP Request/Response. A WebSocket exposes the underlying TCP socket, used in an HTTP request, to the Application layer, and hence creates possibilities for application developers to communicate with the server in a full-duplex manner.

To summarize, most of existing web-based measurement tools proceed using the following steps [57][109]:

— Step 1 (initialisation phase): The client downloads from the web server a container web page that contains the measurement code.

— Step 2: The client probes the web server for a period of time by sending "request" messages, to retrieve an object using a specific measurement approach. The requests are sent using a train of IP packets, the time just before sending the requests is recorded.

— Step 3: Here the web server returns a "response" message to the client, using a train of IP packets (one or more), the time just after receiving the responses is

recorded.

— Step 4: Recorded times are then used to estimate relevant information about the network access performance such as the two-way delay and the available bandwidth.

2.2.3 Network monitoring and anomaly detection

Network performance monitoring is an active area of research encompassing a significant number of techniques. One of its goals is to assess the network state and detect abnormal behaviours deviating from what we consider normal or expected. Performance stability, improvement, or degradation are all possible outcomes of performance monitoring. In order to guarantee network and web performance, service reliability, and Quality of Service (QoS), timely detection of performance anomalies before they trigger unexpected service downtime is critical for service providers. Considerable efforts have been made to address this issue in the literature. Many of these solutions focus on solving the problem in specific domains, by leveraging the power of statistical and machine learning techniques [33, 35, 98]. In fact, a basic network anomaly detection system monitors the performance dynamics of the underlying network and collects the metrics needed to create baseline models or profiles of typical network behaviour [99, 1]. It continuously collects new measurements to track deviations and detect expected or unexpected performance anomalies, then carries out root-cause analysis to identify associated bottlenecks.

Network performance monitoring is challenging mainly because of the scale, heterogeneity and dynamics of today's Internet infrastructure, which leads to the difficulty to clearly distinguish normal instances from abnormal ones, as the boundary between the two is usually imprecise and evolves over time. To enable this distinction, unsupervised learning is more frequently used [19], and consists in clustering the network data according to a set of features, and identifying those clusters that diverge from the dominant clusters representing the normal behaviour. Here in this thesis, we

follow a similar approach of unsupervised machine learning based on data clustering, but this time for the purpose of network monitoring from within the browser.

Data clustering can be defined as a method of analysis that involves studying a set of data that is not labelled and grouping it into coherent and homogeneous subsets (i.e., clusters). To implement such an approach, it is therefore necessary to have a method that allows both to observe and measure the differences between these data and to propose from this analysis a grouping of data into several distinct and consistent classes.

While calculating the differences (and similarities) between data is a relatively easy thing (Euclidean, Manhattan distance etc.), it is more difficult to agree on how to group this data into coherent subsets. We can then distinguish two main approaches [78]:

— Hard clustering: this approach considers that the data must belong to clearly limited subsets; therefore, data cannot evolve between two or more classes.

— Soft clustering: unlike hard clustering, this approach accepts the possibility that data can belong to more than one class. Therefore, we are here on an approach to try to calculate a probability (or a score) to belong to one or more classes.

Following this decomposition, we focus in our work on two popular clustering techniques, K-means [80] and Gaussian Mixture Models (GMM) [81], belonging successively to the hard and soft approaches. For GMM, we consider two variants, the classical GMM variant and the Bayesian GMM (BGMM) [60]. We will give further details on the models later in the coming chapters.

2.3 Web-based network state inference

As highlighted in the Introduction, web-based network monitoring consists of assessing the internet performance from the end-user point of view in a portable and

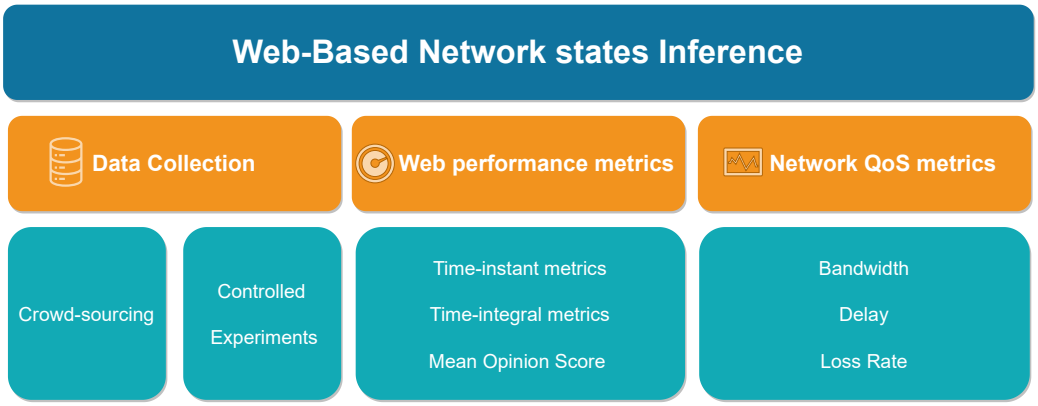


Figure 2.7 – From Web performance to Network Quality

easy way. It is now a popular approach that involves performing measurements from within the browser in an effort to capture the network status as close as possible. From our perspective, this process can be divided into three main steps; (i) the data collection as most of the works related are data-driven, (ii) the web performance metrics, and (iii) the QoS Network metrics. In Figure 2.7, we summarize the typical phases for establishing the link between web and network performance. Each of the following subsections revisits in detail the different phases.

2.3.1 Data collection

Several recent studies have been carried out to estimate the quality of web browsing [79, 49, 3, 30, 10]. These studies can be divided into two broad classes.

2.3.1.1 Crowd-sourcing techniques

The first class of techniques are based on crowd-sourcing, which consists in collecting data from a large set of real users encountering real network conditions [34, 95]. Network conditions being driven by real scenarios, these techniques do not explore unrealistic areas, where for example losses are too high or bandwidth is too large. Some extreme scenarios might exist but given their low probability, they tend to dilute in the mass of data corresponding to common scenarios. In general, these

techniques should face the heterogeneity of users resulting from differences in terminal capacities, which is not necessarily known, the difference in access networks, whether or not their traffic is throttled, etc., all this means that the measurements taken by crowd-sourcing are biased by various factors that are not really controllable. Fortunately, the large number of users can reduce this measurement bias. On the positive side, and in addition to capturing the real scenarios, these techniques allow detecting scenarios that someone might not think of beforehand.

2.3.1.2 Controlled experiments

Second-class techniques are based on the construction of a model (or several models) for web quality based on datasets collected by controlled experiments [84][50][4]). Unlike the previous class, the approaches of this class, since they do not necessarily know the network conditions that users may face in reality, must widely explore the different possible network conditions. Several methods have been proposed to effectively explore the very wide space of possible conditions, such as the quasi-Monte Carlo method [54], the active learning method [51] or the Fourier Amplitude Sensitivity Test (FAST) [88] that we are exploring in this thesis. Note here that contrary to existing approaches that focus on estimating web quality based on network conditions, our approach targets the estimation of the network performance itself, leveraging the web measurements. The advantage of this approach stems from the fact that Web measurements are of passive nature and are available at very low cost, at the user terminal inside the browser, whether mobile or fixed. If proven to capture the network conditions, they can avoid overloading the network with active measurements (i.e., traffic injection), source of measurement bias and consumers of CPU and data at the access.

2.3.2 Web performance metrics

As we mentioned before, choosing the web metrics that faithfully reflect the end-user Quality of Experience (QoE) is challenging. In this section, we revisit this topic, starting by defining the term Quality of Experience and then presenting the state of the art of the different web metrics that reflect it.

2.3.2.1 Web quality of experience

In general, Quality of Experience (QoE) refers to the level of satisfaction of the user with a network service such as web browsing. It is necessary to differentiate what Quality of Experience represents for a user from how it is actually measured. Indeed, the QoE represents all the objective and subjective characteristics specific to satisfying, retaining or giving confidence to a user through the life cycle of a service. In contrast, the measurement of QoE is done by a subjective evaluation of a person or a coherent population of users on a service they use.

The official definition of QoE is given by the recommendation P.10/G.110 of the International Telecommunications Union states that [13]

> "*Quality of Experience (QoE) is 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.*"

The Web Quality of Experience denotes the quality of web users' experience (WebQoE).

2.3.2.2 Perceived performance metrics

Subjective approaches usually require direct end-user involvement, and they are widely used to quantify the perceived QoE for networked applications such as the web. Numerous publications using these methods have been produced [18, 56, 85]. The most widely used method for subjective tests is the Mean Opinion Score (MOS).

MOS is a user opinion-based metric standardized in an ITU-T recommendation [94]. It has five levels from 1 to 5, where 1 represents bad quality and 5 represents excellent quality. Each level is meant to reflect users' judgment of the Web service quality. However, using MOS to represent the QoE has many shortcomings [41], such as:

— High set-up cost.

— Very time-consuming set-up and production.

— Inappropriateness to be used in real-time.

— Lack of repeatability.

2.3.2.3 Web objective metrics

In order to overcome the limitations of user opinion score-based metrics, researchers have suggested objective metrics to quantify the perceived quality of experience from measurements performed from within the web browser. For example, several studies have pointed out the impact of the latency on WebQoE: the lower the latency, the higher the WebQoE [25]. The loading of a web page involves a long list of events encompassing the request of the page, the response of the server and the rendering of the downloaded content. WebQoE is far from being a simple function of latency, often requiring to monitor the loading in all its finite steps. As a result, the literature introduces many types of other sophisticated web metrics that can be easily calculated and that are better related to WebQoE.

In this section, we present the most prominent objective web performance metrics. In particular, we have two main categories: time-instant and time-integral measures [11].

Time-instant Metrics. Time-instant metrics are easily computed since they do not require any user interaction and are measured directly from within the browser. Indeed, time instant metrics are basically related to the web page loading process, specifically when an event occurs, and they are obtained using the browser navigation timing information. The PLT (Page Load Time) is one of the commonly used

metrics [26, 102, 72, 103]. However, recent studies deduce that this metric alone cannot give a precise estimation of web browsing quality, hence the need for finding more suitable metrics that are closer to screen rendering and its subjective evaluation by the user [20]. Table 2.2 summarize the most used metrics that express user's WebQoE.

Table 2.2 – Common web performance time-instant metrics

Metric	Definition
TTFB	When the first byte of the payload of the web page arrives.
DOM	When the Document Object Model (DOM) tree is parsed.
FP	When the first pixel is rendered on the screen (the first visual change).
TTI	When the web page is able to respond to the user interaction.
ATF	When the web page renders the contents within the above the fold area.
PLT	When all the objects are downloaded and the web page is rendered.

Time-integral Metrics. Metrics belonging to this category take into consideration all the events in the web page loading process, and as in the name, they are computed by integrating the time until the last event in the web page waterfall is triggered. In particular, Google proposed the Speed Index [40], which expresses the time until the web page is rendered in the screen's visible area or what we call the Above The Fold field [39]. However, measuring the Speed Index is very challenging since it is computationally intensive and not large scale oriented. Table 2.3 illustrates some of the time-integral metrics.

In this thesis, we opt for measuring the maximum of timing components related to navigation, which are well presented in W3C's specifications 'Chrome Navigation Timing API' [64] and the 'Paint Timing API' [73]. So we consider the following:

Table 2.3 – Some web performance time-integral metrics

Metric	Description
Speed Index [40]	Integral time related to the visual progress of the Web page
Ready Index [67]	When objects became ready in the visible area of the Web page
Byte Index [11]	Integral time related to byte level completion of a web page

DNS, the time when finishing looking up for the domain. *Connect Start*, the time to start the connection with the server. *Request*, the time when sending the request to a server to retrieve a resource. *Response*, the time when receiving the last byte of the response. *DOM*, the time when the load of the document object model finished. *First Paint (FP)*, the time when the first pixel is rendered. *First Contentful Paint (FCP)*, the time when the first bit of content is painted. Finally, the *Page Load Time (PLT)*, the time when the web page finishes loading. We refer to Figure 2.8 for a chronological illustration of these metrics.

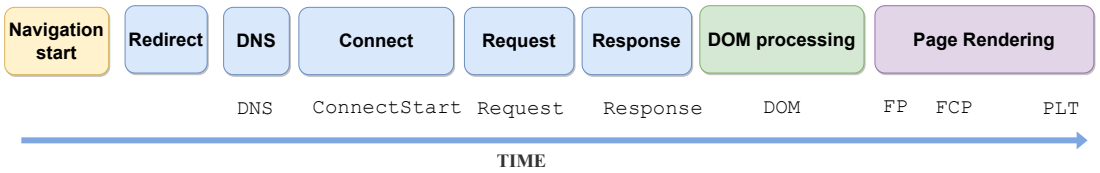


Figure 2.8 – Web browsing main events and related metrics according to W3C's specifications

2.3.3 Network QoS metrics

Before planning how to collect data on the network and the service traffic, it is important to identify which metrics are necessary to infer the network quality.

A network performance metric is the basic metric of performance specific to the underlying network; several metrics are proposed in the literature [22][46][63]. How-

ever, according to [37], the most relevant network performance metrics can be defined into four types: Availability, Loss, Delay and Utilization.

Availability means the functionality as well as the connectivity of the network elements. Loss represents the percentage of packets lost between the sender and target during a specific time interval. Delay metrics can measure the one-way delay, as well as the RTT, the Round Trip Time between the host and the target. They can also assess variation in the transmission latency or jitter. Utilization stands for the end-to-end throughput of the link and is expressed as a percentage of the access rate. Several works consider these network parameters to study network performance, especially latency and bandwidth, such as in [83][53][23][5], etc.

Taking into consideration the asymmetric nature of web traffic and the fact that it is primarily download traffic, we have identified three metrics to collect in this thesis. One of these metrics is the Round-Trip Time (RTT) [110] which is the latency necessary to communicate from one host and back to it through our final destination. Then, we have the Download Loss Rate [106], which is the percentage of packets lost in the download direction (i.e., the number of packets lost over the total number of packets sent). Finally, one can find the Download Bandwidth [32], which stands for the maximum end-to-end throughput in the download direction.

Chapter 3

Network performance inference form within the browser

3.1 Introduction

Web browsing remains one of the dominant applications of the internet, so inferring network performance becomes crucial for both users and providers (access and content). Indeed, it allows content providers to ensure a good quality of their services by identifying the root causes of service degradation. Also, it gives the end-users a better understanding of the performance they have (state of the networks). A widely used monitoring technique involves performing measurements from within the browser to capture the network status as close as possible; we talk about web-based network monitoring. Consequently, several network troubleshooting tools have seen the light recently, e.g., NDT [66], MobiPerf [61], SpeedTest [83], and Fathom [23]. Yet, these tools are either computationally expensive, less generic or greedy in terms of data consumption.

Bridging the gap between web performance and network performance requires deploying measurement tools at both levels (the web and the network) to build models relating both of them. Once these models built, network performance can thus be estimated from web performance measurements without the need to access or probe the network; the latter measurements are able to be carried out efficiently and passively within the browser, without extra cost on the user terminal and the network itself.

So the main purpose of this chapter is to leverage passive measurements and machine learning techniques (ML) to infer the main properties of the underlying network (e.g., delay, bandwidth, and loss rate) from web performance metrics (e.g., Connect Start, Page Load).

In order to enable this inference, we propose a framework based on extensive controlled experiments where network configurations are artificially varied, and the web is browsed. ML is then applied over the collected data to build models that estimate the underlying network performance. In particular, we contrast classical ML techniques (such as random forests) to deep learning models trained using fully connected neural networks and convolutional neural networks (CNN). By doing so, we realize two main objectives: (i) reducing to zero the cost of active measurements and (ii) inferring the characteristics of the path to the web server origin of the web page without the need for the collaboration of a dedicated measurement server, as is the case with existing tools.

Overall, the contributions of this chapter are :

1. We engineer a methodology for collecting a large dataset that links web performance measurements to underlying network measurements over an example of three network metrics, namely the round-trip time (RTT), the download bandwidth and the loss rate.

2. We present our convolutional neural network (CNN) to calibrate data-driven models that allow network performance inference as accurately as possible. Our deep learning models are calibrated using a controlled experimentation approach. We artificially change the network conditions and automate a web browsing activity; then, we use the ground truth on the network state together with the passive measurements available in the browser for the training and validation of our models. We prepare the ground for this modelling by carrying out a sensitivity analysis to understand the dependency between the performance at the web level and the one of the network.

3. We validate our solution and study its efficiency by comparing it with two well known ML approaches: Neural Networks and Random Forests.

The rest of this chapter is organized as follows. In Section 3.2, we describe in detail our approach for estimating the underlying network metrics from web performance measurements, as well as the data collection process. Then, we present our convolutional neural network (CNN) to calibrate models that allow network inference as accurately as possible. Later in Section 3.3, we evaluate our approach and discuss the results of our experiments. Finally, we summarize in Section 3.4 the main contributions of this chapter.

3.2 Estimating network status from web performance measurements

3.2.1 Methodology

In order to predict network status departing from web measurements, the first step is to collect a dataset that captures the link between network QoS metrics and web QoS metrics. We proceed by extensive controlled experiments where network configurations are artificially modified and measurements of both network and web browser are collected. Then, we apply data analysis techniques to estimate the network status from the browser-level measurements. For that, we propose a distributed system based on different entities that provides a platform to link the input (underlying network metrics) to the output (web performance metrics), see Figure 3.1 for more details.

In our system, the *Experimenter* unit communicates directly with the *sampler*, using the `GetSample()` function, which requests the next configuration to experiment. The configurations consist in tuples of RTT, download bandwidth and download packet loss rate. It then enforces these configurations thanks to a *Network Emulator*,

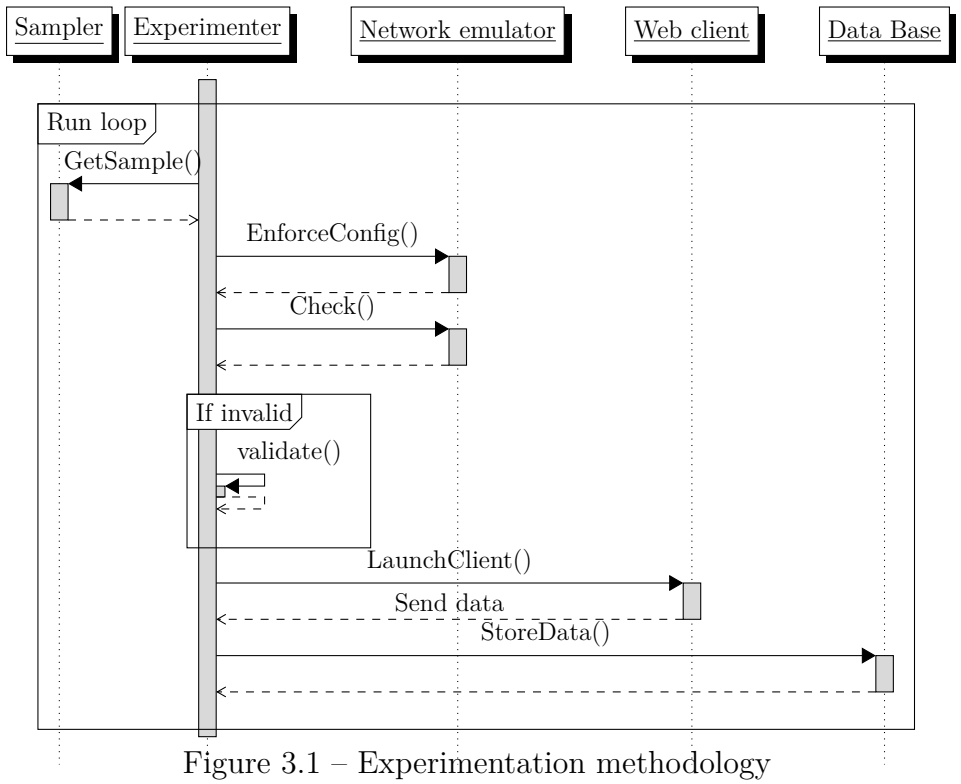


Figure 3.1 – Experimentation methodology

using the `EnforceConfig()` function.

Our approach is based on lab experiments, where we aim to have under our control the network conditions between the client and the real web servers. Whereas *tcconfig* [89], which we used to enforce network configurations, allows controlling the access network, it does not provide control over the entire network path to the web server. On one side the Internet Service Provider is out of our experimental network and on the other side the experimentation system requires performing a real page loading from a cloud that is again out of our control. It follows that *tcconfig* is not always able to enforce the wanted network conditions (e.g., lower bandwidth or larger delay than needed). In order to handle the noise coming from the uncontrolled part of our experimental setup, we integrated measurement tests (using the `Check()` function) to ensure the validity of the samples. In particular, for each desired configuration, we implement:

1. A TCP throughput test to ensure that the available bandwidth is larger than

the one we want to enforce.

2. RTT and loss rate noise estimation tests.

Test 1 is performed before enforcing the network state by downloading the web page directly from the target server and measuring its download rate. Then we check if this value is higher than the one we want to enforce.

Test 2 is necessary to deduce the network delay from the emulated one. For example, if the expected RTT is 1000ms and the measured one is 50ms, the final configuration will be 950ms. This test is implemented by probing web pages using the ping tool.

Once the network configuration has been validated, the *web client* is launched, using the `LaunchClient()` function. Our developed web extension monitors for each visited web page the different web metrics (Connect Start, DNS, Request, Response, DOM, FP, FCP, PLT), as well as other web page characteristics, such as the number of objects, the size, and the protocol supported. At the end of the experiment, the results are retrieved, and the resulted statistics are stored by the *Experimenter* using the `StoreData()` function.

Note that we collect statistics from the 100 top popular web pages according to the Alexa ranking [1]. We will give more details on the implementation of our platform later in the text. The dataset obtained applying our methodology is composed by $10,000$ experiments for each of the 100 top popular web pages. These experiments correspond to 10 repeated downloads of each page under the same network conditions, for a number of different network conditions equal to 1000 obtained using the FAST method defined in Section 3.2.2 [88].

3.2.2 Sensitivity Analysis

In order to have a clear understanding of the correlation between the measured web metrics and the enforced network configurations, we start our study with a

1. `https://www.alexa.com/topsites`

sensitivity analysis. This allows to better see the interplay between the network and the web browsing performance. The idea here is to reveal the degree of dependence between both of them, and consequently whether the combination of different web metrics can bring information about the underlying network metrics.

To do so, we consider the Fourier Amplitude Sensitivity Test (FAST) [88], one of the most widely used sensitivity analysis techniques. FAST implements a periodic sampling strategy based on appropriate frequencies (one frequency per dimension) to ensure good coverage of the area to be sampled. In addition to providing a sequence of input tuples to experiment with, the method also allows assessing the sensitivity of the output labels (i.e., web metrics in our case) to the input metrics (i.e., network metrics in our case) through the analysis of the spectrum of the obtained labels.

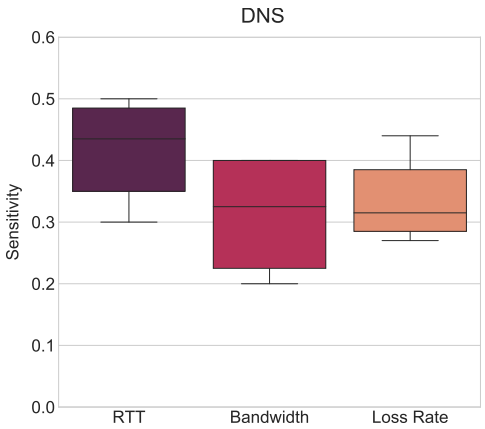


Figure 3.2 – DNS sensitivity to network QoS

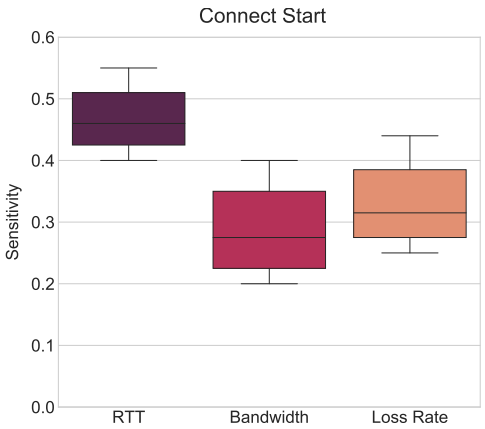


Figure 3.3 – Connect Start sensitivity to network QoS

Although this technique does not consider the real properties of the network model in question here, it allows exploring the space of possible network configurations without missing important points. Unlike a generation of random samples based on the Monte Carlo method [54]. This technique allows covering, indeed the space of network configurations efficiently by avoiding repetitions, thanks to the variation of frequencies. The partial variances obtained in this way make it possible to see the contribution of network metrics to the overall variation measured. The partial

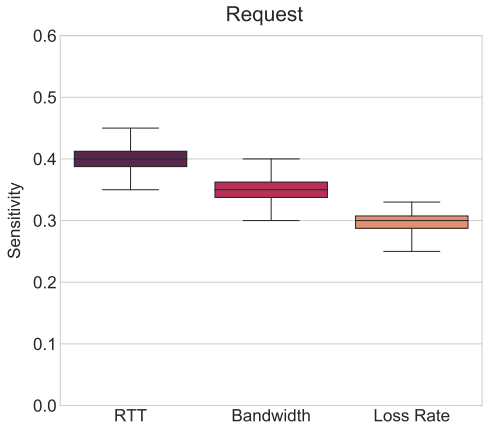


Figure 3.4 – Request sensitivity to network QoS

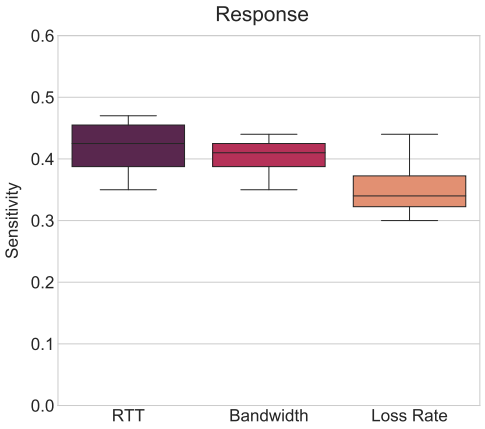


Figure 3.5 – Response sensitivity to network QoS

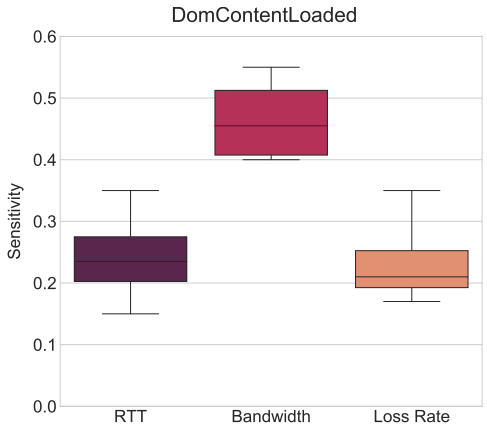


Figure 3.6 – DOM sensitivity to network QoS

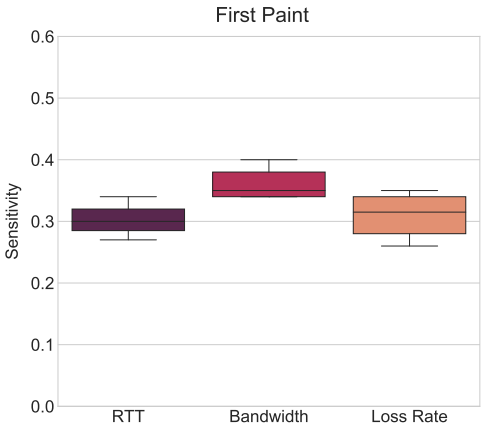


Figure 3.7 – FCP sensitivity to network QoS

variance of an input metric corresponds to the partial energy of the output label summed over all frequencies multiple of the characteristic frequency of the input metric. The total variance (or total energy) is the sum over all frequencies. The ratio of partial variance to total variance (being a real number between 0 and 1) models the sensitivity of the output label to the input metric. One can see this as the participation of the input metric, by its variability, to the total variability of the output label.

Sensitivity indices are calculated using ANOVA-like decomposition of the function for analysis based on a variance-based method. Suppose the function is $Y = f(X) = f(x_1, x_2, ..., x_n)$ where x_i are input model parameters and Y is the output. The sensitivity index of the input metric x_i is then defined as the normalized conditional variance:

$$S_{x_i} = \frac{V_i}{V(Y)}. \tag{3.1}$$

Where $V(Y)$ represents the variance of model output Y resulting from uncertainties in all input model parameters, such that;

$$V(Y) = \sum_{i=1}^{n} V_i + \sum_{i<j}^{n} V_{ij} + \cdots + V_{12...n}$$

V_i is the marginal variance of x_i given by :

$$V_i = \mathrm{Var}\left(E(Y \mid x_i)\right)$$

And V_{ij} is the contribution of the interaction between x_i and x_j:

$$V_{ij} = \mathrm{Var}\left(E\left(Y \mid X_i, X_j\right)\right) - \mathrm{V}_i - \mathrm{V}_j$$

In general, $V_{i_1 i_2 ... i_r}$ is the cooperative fractional variance of order r.

In our case, we obtain for each web metric and for each web page, three sensitivity

indices (S_{RTT}, $S_{Bandwidth}$, and $S_{LossRate}$). The boxplots (see Figures 5.9a, 5.9b, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9) display the dispersion of sensitivity indices over web pages for each of the eight web performance metrics. The y-axis in the figures shows the sensitivity index. We notice a strong correlation between input and output features, with some particular behaviors. In particular, Connect Start, DNS and Request are more sensitive to delay. Response is more sensitive to delay and bandwidth. First Paint and First Contentful Paint are more affected by bandwidth. PLT, is sensitive to all the three network metrics, and we observe that their impact is closely the same

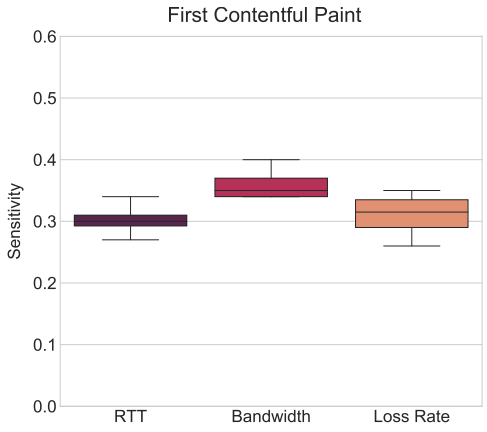


Figure 3.8 – First Paint sensitivity to network QoS

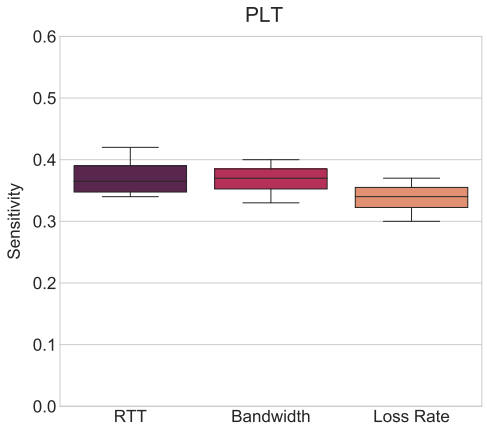


Figure 3.9 – PLT sensitivity to network QoS

From these results, we can conclude that the patterns between network and web metrics are complex, thus the need of advanced models if we want to estimate network status from a specific web metric combination.

3.2.3 CNN-based network performance estimation

In this section, we give a detailed overview on the model we opt for to perform the estimation of network conditions from web measurements. We justify the use of such model later by comparing it to other machine learning models.

A Convolutional Neural Network (CNN) is a deep neural network topology that typically combines convolutional filter layers in conjunction with a fully connected

neural network. A CNN works well for extracting more detailed features within the data, which will then be used to form more patterns within higher layers. Based on this advanced technique, we aim to build a model that estimates the network status by performing a CNN driven regression analysis, considering as input the web performance metrics (Connect Start, DNS, Request, Response, DOM, FP, FCP, PLT), and as output the estimated value of the underlying network metric (delay, bandwidth, or packet loss). Typically, we will have three regressions, one for each of the network metrics we consider in this work.

We start by processing the obtained data needed to fit the estimator, then we separate our dataset into training and test sets; for that we pick 70% of the dataset randomly as training set, and we consider the rest as test set. We use the training part to calibrate our CNN model and the test part to validate its performance. Our CNN estimator consists of: (i) an input layer, where we have 11 elements (i.e., the considered web metrics), (ii) a first 1D CNN layer (one dimension CNN layer), (iii) a second CNN layer, (iv) a Max pooling layer, (v) and finally the fully connected neural network layers.

Regarding the hyper-parameters to tune, the most important ones are the kernel size N_s, the number of kernels (or filters) N_k and the number of hidden neurons inside the fully connected network part N_n. For the convolutional layers, we define 100 filters (also called feature detectors), so we take $N_k = 100$ with a kernel size $N_s = 3$, this allows us to train 100 different features on the first layer of the network. For the fully connected feed forward neural network, we consider two hidden layers with the same number of neurons. The optimal number of neurons for these layers is studied in the following section. In the hidden layers, we use the activation function ReLU, which is a non-linear activation function.

We train our model using the ADAM optimization algorithm [52]. This learning method is very effective since it converges fast and provides good results compared to the classical gradient descent approach. Furthermore, as training progresses, the

ADAM algorithm computes a loss function, the back-propagation of this function modifies the network in order to minimize the loss which by the end leads to optimal estimator weights.

In order to evaluate the performance of the obtained CNN model, we calculate the Mean Absolute Percentage Error (MAPE) over the test set. We target the estimation of each of the network conditions given the collected web measurements and the pages' features. We get an estimation of network performance for each visited web page.

$$MAPE = 100\% \frac{1}{n} \sum_{i=1}^{n} \mid \frac{Estimated\ Value - Real\ Value}{Real\ Value} \mid \tag{3.2}$$

3.3 Performance evaluation

3.3.1 Platform implementation

Having introduced the general methodology before (see Figure 3.1), we now focus on its implementation. In particular, we have the Experimenter that executes the tests and starts the experiments by communicating with all the other entities. This Experimenter is composed of four parts. The first part, developed in Python as a simple Finite State Machine (FSM), sets the network conditions and generates network configurations to experiment with. These configurations are enforced in the network through the network emulator *tcconfig* [89]. The second part is built as a web page that uses the Service API to control the experiment and collect statistics. The web client is composed of two entities: the browser (we use Google Chrome) and the extension; the browser is in charge of loading the web pages while the extension is a plugin developed in JavaScript to perform measurements based on *Chrome Navigation Timing API* and the *Performance Navigation API* [73, 64], both being W3C recommendations. The resulting data is then retrieved by the Experimenter and stored in a database.

3.3.2 Results

Here, we compare the performance of our CNN-based network estimator with two ML approaches: Fully Connected Feed Forward Neural Network (NN) and Random Forest (RF).

Hyper-parameter tuning of the estimators In order to fine-tune the hyper-parameters of the proposed estimator and the models with which we compare our solution, it was necessary to use various settings. For neural network models (i.e., CNN and NN), we opted for the Keras platform [2] to evaluate the estimation efficiency. We considered models of neural networks with two hidden layers, for which we varied the number of neurons from 100 to 500 neurons in steps of 100. We had as input the web metrics (Connect Start, DNS, Request, Response, DOM, First Paint, First Contentful Paint, PLT) plus other page features (Global size, Protocol, Number of objects) and the output is the estimated network metric (one model per network metric).

Fig. 3.10 shows the scatter plots of the estimated packet loss rate as function of the real one respectively for 100, 200, 300 and 400 neurons and for both NN and CNN. The same results were observed for the bandwidth and RTT metrics. We can see how the performance of both NN and CNN increases significantly when we add more neurons. Particularly, we obtain a higher accuracy when we reach 400 neurons. Beyond this value, the accuracy starts to decrease. One can also note that CNN clearly outperforms NN in predicting the packet loss rate.

The second model that we assessed is Random Forest (RF), which is a regression technique that consists in calibrating and combining multiple decision trees to create a more powerful model. To build the RF model we used the scikit-learn library [3]. The number of trees is a main parameter of RF. We changed this number from 100 to 700 in steps of 100 to study its impact. The metric Mean Absolute Percentage

2. `https://keras.io/`
3. `https://scikit-learn.org/`

(a) Estimated loss rate versus real one for NN

(b) Estimated loss rate versus real one for CNN

Figure 3.10 – Comparison between NN and CNN using a different number of neurons

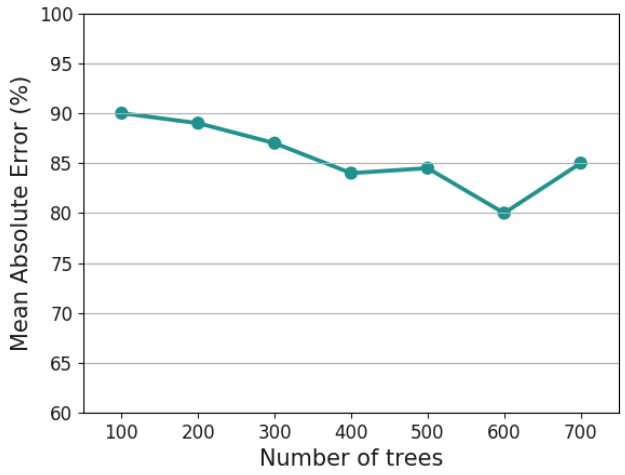


Figure 3.11 – RF performance versus number of trees for Loss Rate

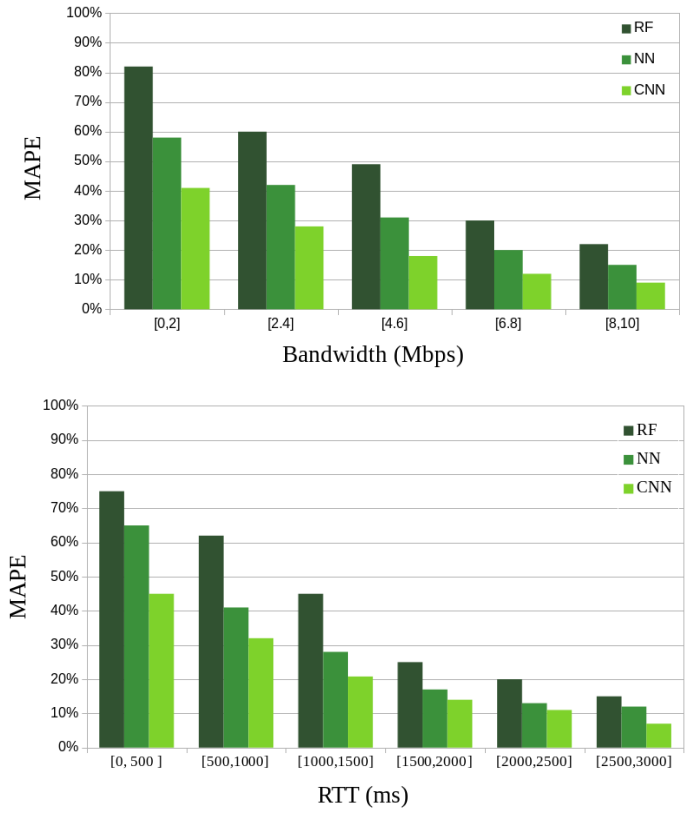


Figure 3.12 – CNN against NN and Random Forest

Error (MAPE) is plotted versus the number of trees for the Packet Loss Rate in Figure 3.11. The lowest error was observed for 600 trees, we choose thus this setting for RF in the rest.

CNN versus NN and RF Now we compare the three ML techniques using their best tuning found above. Figure 3.12 compares them to each other in terms of MAPE for different ranges of network QoS metrics. Random Forest, despite its known power, shows the largest estimation error, which can go up to 80% for low ranges. This illustrates the difficulty of the task. Fully Connected Neural Networks come next, then they are followed by CNNs, which show the best estimation accuracy. It can also be observed how for the three techniques, the relative accuracy improves when ranges get higher. In particular, when using CNNs, the error drops to less than 10% for a download bandwidth around 10Mbps.

3.4 Conclusion

We presented in this chapter an efficient and novel method to estimate network performance metrics from web metrics that can be collected passively and easily from within the browser. We developed a platform to collect our own dataset and designed a methodology around deep learning for network estimation and the FAST method for sensitivity analysis.

Our results underlined the difficulty of the task, given the complexity of the relationship between the network and the details of the rendering of a web page. Only Convolutional Neural Networks were able to provide acceptable results that can get as accurate as a few per cent for some ranges of the underlying network metrics.

In the coming chapter, we revisit the topic of network performance measurement from within the web browser. We study the impact of the web page complexity on the estimation of the network states. To that aim, we capture specific web measure-

ments that we extend with information on the characteristics of the visited web pages (e.g., web page size, number of objects, Protocol supported, etc.). Then we study the efficiency of our approach by comparing it experimentally with other network troubleshooting solutions.

Chapter 4

Impact of web page complexity on network performance inference

4.1 Introduction

In the previous chapter, we highlighted through experiments the efficiency of our method in estimating network performance metrics from web metrics, using Machine Learning, and Convolutional Neural Networks in particular. For that, we leveraged end-to-end web measurements that can be easily and passively captured from within the browser without injecting additional measurement traffic. Our approach allows us to estimate the end-to-end path performance to the actual web server without having control of this letter, as required by existing measurement tools. However, the loading process of a web page is still an essential factor that we must consider in our estimation models. In fact, today's web pages incorporate plenty of objects fetched from multiple servers through multiple connections and use complicated rendering technologies with advanced underlying protocols such as HTTP 1.1 and HTTP/2. This complexity raises tow main concerns on two different levels:

— **The web page level:** in terms of the number, the type, and the size of the objects, and the page itself, as well as the processing of the objects and the rendering to the browser.

— **The transfer protocols:** As we mentioned before HTTP has evolved [43] from the first HTTP/0.9 to HTTP/1.1 [31] with many important extensions

namely HTTP over TLS or HTTPS [74] (the secured version of HTTP). Then, based on Google's SPDY protocol, HTTP/2 has seen the light and attempts to tackle the inefficiencies of HTTP/1. Indeed, HTTP/2 includes the advantages provided by SPDY and adds its own optimization approaches. The network estimation can thus depend on which transfer protocol is used.

In this chapter, first, we highlight the impact of these two aspects on estimating two specific network metrics, the delay, and the download bandwidth. Then we compare the accuracy of our estimation model with the most known monitoring solutions. As in the previous chapter, we follow a controlled experimental approach to derive our inference models and to validate them. In more detail, the main contributions of this chapter are:

1. Considering the influence of the loading process of web pages, we study and examine this impact on our estimation models, for that we capture web-specific measurements with information on characteristics (e.g., web page size, number of objects, etc.) of the visited web pages as well as the protocols supported (e.g., HTTP/1.1, HTTP/2, HTTPS, etc.).
2. We train and calibrate a Convolutional Neural network model (CNN) to estimate the network state from these passive measurements without probing the network.
3. To compare with existing web-based monitoring solutions, we propose an integrated framework where we implement our solution and mimic the behaviour of other solutions to ensure a fair comparison between the different approaches in a fully controlled environment.

The rest of this chapter is organized as follows. In Section 4.2, we describe our approach for training the deep learner model and for collecting the required data. Next, in Section 4.3, we present the results related to the impact of the web page complexity on our estimation models. Then, in Section 4.4, we highlight the efficiency of our approach. In particular, we present the integrated platform used to compare

our solution with existing solutions, as well as the results of our experiments and our main observations. Finally, we conclude the chapter in Section 4.5.

4.2 Delay and bandwidth inference

4.2.1 Data collection phase

The same as before, we proceed by extensive controlled experiments, where network configurations are artificially modified and measurements on both levels of network and web browser are collected. For that purpose, we use the distributed system described in Figure 4.1 based on different entities to provide a platform linking the input (underlying network metrics) to the output (web performance metrics). Next, a description of these entities.

The "Experimenter" generates network configurations and enforces them within the network (Delay and Bandwidth) – Step 1 –, then launches the web client – Step 2 –. This entity is in charge of returning a valid sample. It is in part developed in Python, as a simple Finite State Machine (FSM) that sets the network conditions, and in part it is built into a web page, to control the experiment (in our use case the loading of web pages).

The "Web Client" is composed by two entities: the browser (in our case Google Chrome) and the extension. The browser is responsible for loading a specific web page, while the extension developed in JavaScript collects all the information that we want using "Chrome Navigation Timing API" and the "Performance Navigation API", which are w3c recommendations. Finally, the collected data are stored in – Step 3 –.

As in previous chapter, network configurations are generated using FAST method (Fourier Amplitude Sensitivity Test [88]). All the traffic generated by the service goes through a network emulator that enforces the network status when needed. The web page and the experimenter communicates through a Web socket via custom

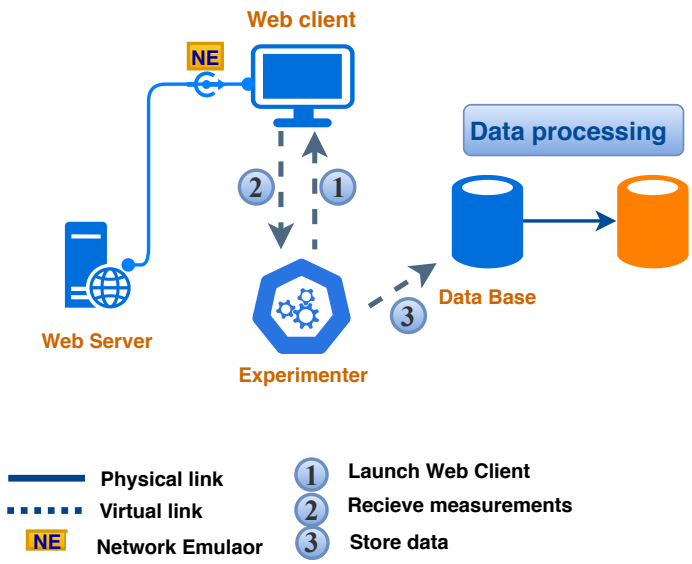


Figure 4.1 – Data collection and processing phase

commands. It has to be pointed out that all the local traffic is not affected by the network emulator, only the traffic between the browser and the web server is managed. The data set obtained applying our methodology is composed by 5000 entries for each of the 500 top popular web pages according to Alexa ranking [1].

Before starting the estimation phase, we should first be sure that the data set fits the estimator well. To do so we perform a data cleaning by removing missing and erroneous values, then we perform a data transformation that considers as input the web performance features and as output the network performance metric (i.e., RTT or bandwidth). Finally, we split our data on training and test sets; for that purpose we pick 70% of network scenarios and 80% of web pages randomly as training set, we consider the rest as test set.

4.2.2 Estimation phase

Based on the results of Chapter 3, it can be seen that CNN gives the highest accuracy compared with traditional machine learning techniques, such as Random

1. `https://www.alexa.com/topsites`

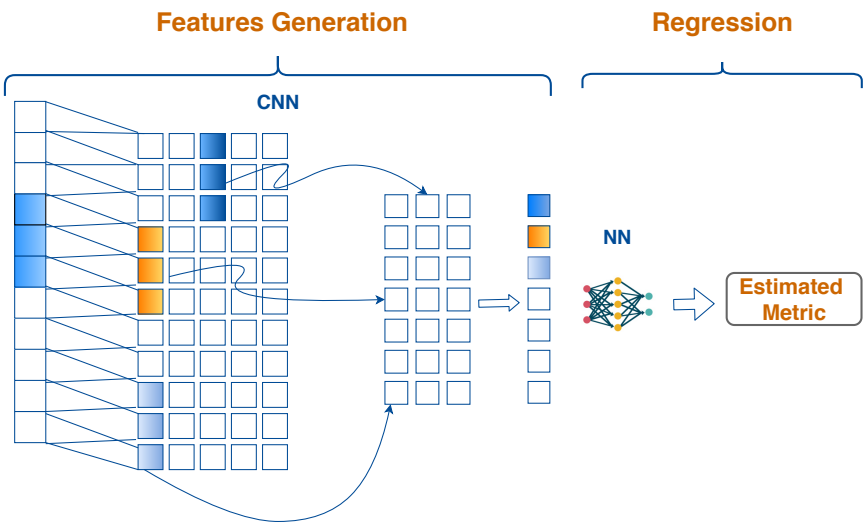


Figure 4.2 – Estimation phase

Forest. We thus opt for this deep learning technique to build a model that predicts the underlying network state, considering as input web metrics of a page together with information on its content, and as output the estimated network metric: RTT or download bandwidth (as summarized in Table 4.1).

Basically, we have two regression problems, for each network metric. As shown in Figure 4.2, in the estimation phase, we build the CNN based model consisting on two 1-dimensional convolutional layers, with a number of filters equal to 100, and a kernel size of 3. This is then followed by a Max pooling layer. Lastly, we have two hidden layers of a fully connected neural network with 400 neurons each. We use ReLU as an activation function, which is a simple non-linear activation method. We do not use any activation function for the last layer, since we are in a regression scenario, and so we are rather interested in estimating raw values without transformation. Finally, we use the ADAM algorithm during the training phase in order to optimize the loss function [52]. Then, we calculate the Mean Absolute Percentage Error to evaluate the performance of our model.

Table 4.1 – Web and network performance metrics

Network metrics	RTT – round trip time –
	Bandwidth – maximum end-to-end throughput –
Web metrics	Connect Start
	DNS
	Request
	Response
	DOM
	First Paint (FP)
	First Contentful Paint (FCP)
	Page Load Time (PLT)
Page characteristics	Page Size (Size)
	Number of objects (NumObjects)
	Protocol supported (Protocol)
	Median of objects size (Median Objects)
	Total objects size (Objects' Size)
	First Quantile of object's size (Q1 Objects)
	Third Quantile of objects size (Q3 Objects)
	Maximum objects size (Max Objects)
	Minimum objects size (Min Objects)

4.2.3 Feature importance study

Feature selection based on feature importance is an important step in deep learning algorithms, as it brings a huge impact on the accuracy of the model. Indeed, this study brings several advantages such as reducing over-fitting, increasing accuracy as well as reducing the training time.

Many techniques can be used to perform this analysis, one way is the use of the importance feature property in Random Forest regressors. Random Forests are most often used for estimation purposes, but they also have the ability to know which feature has the most effect in building the model. Depending on the output of the training phase, the use of the feature importance capacity of Random Forests gives a score for each feature of the data. The higher is the score, the more important or relevant is the feature for the estimation of the output variable. We apply this method on our dataset by building a random forest model. For that, we consider all

the input features (web performance metrics plus page characteristics, see Table 4.1). Figure 4.3 shows what are the most important features in our dataset.

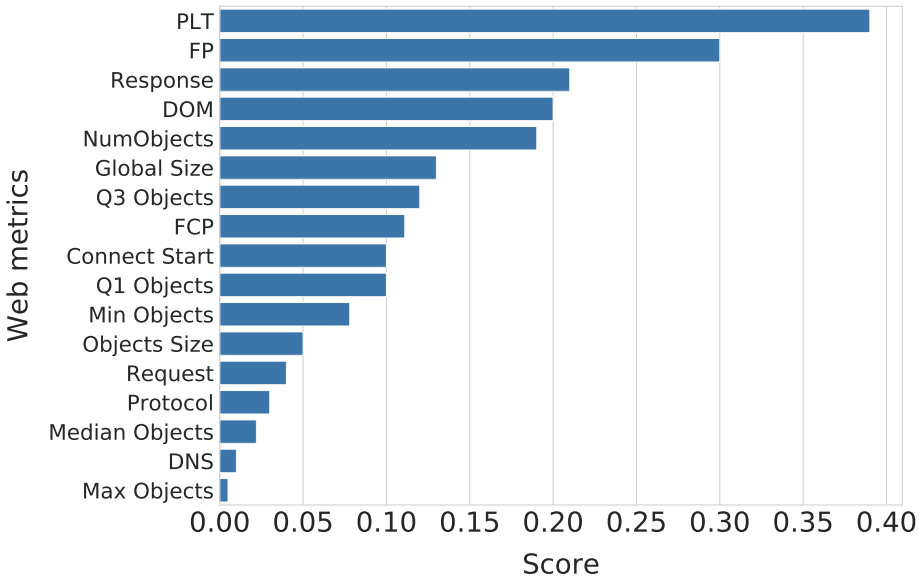


Figure 4.3 – Feature importance using Random Forest

Another way to select features is to use the Correlation Matrix. This technique shows how the features are related to each other, features with high correlation with each other have more likely the same effect on the target variable, hence we can use one of them. We plot in Figure 4.4 the correlation heat map of features in our dataset using the Pearson correlation coefficient [93]. The Pearson correlation coefficient of two variables X and Y is defined as their covariance divided by the product of their standard deviations, which gives values between -1 and 1. The smallest the coefficient in its absolute value, the less the linear correlation between the two variables. The values -1 or 1 correspond to a total correlation between the two variables, and the value 0 to a non-linear correlation.

Based on the results above, and the accuracy of the obtained model, we removed the following web metrics: First Contentful Paint since as shown in the heat map it is strongly correlated with First paint metric, and the last feature in Figure 4.3, the maximum size of page objects.

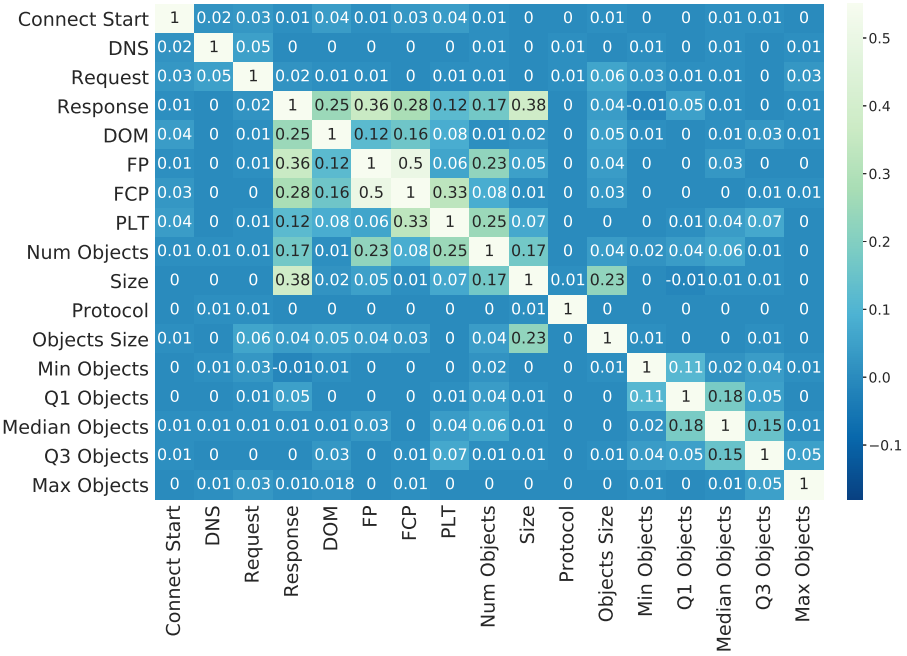


Figure 4.4 – Heatmap correlation matrix of features

4.3 Performance evaluation

As we previously stated, the loading process of a web page is an essential factor that we must consider in our estimation models. Thus, in this section, we present the results related to the impact of the web page complexity and the transfer protocols on the accuracy of inferring the network state.

4.3.1 Impact of web page size and number of objects

Web pages are key elements of network estimation framework. In order to study their impact on the accuracy of the estimation, we plot in Figure 4.5 the global estimation accuracy for 500 web pages for both RTT and download bandwidth, as well as box plots in Figure 4.6 displaying the RTT and download bandwidth estimation error as a function of the number of objects for different web page sizes. The y-axis in all the figures shows the mean percentage error given by Equation (4.1).

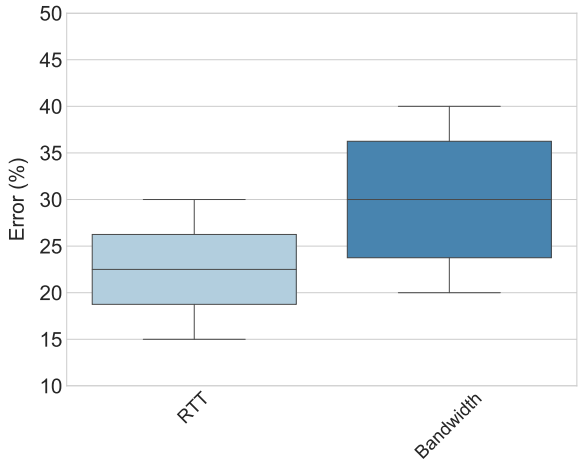


Figure 4.5 – RTT and bandwidth estimation error for the top 500 web pages

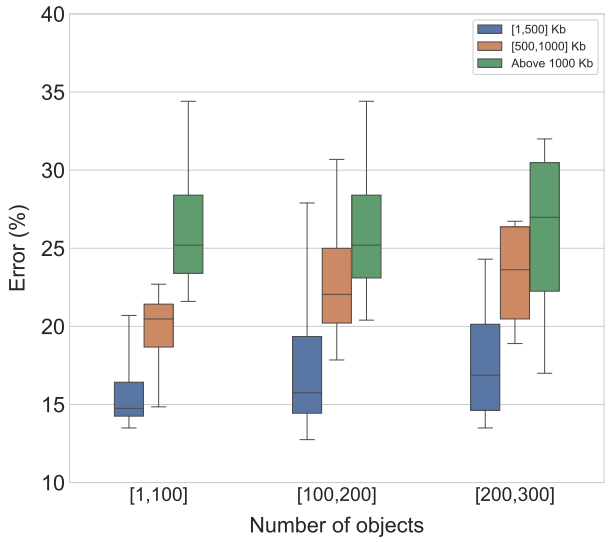
$$Error = 100\% \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Estimated\ or\ Measured\ Value - Real\ Value}{Real\ Value} \right| \tag{4.1}$$

Figure 4.5 shows that the median estimation error does not exceed 22.5% for RTT and 30% for Bandwidth. And according to Figure 4.6, the accuracy of estimating RTT improves significantly for web page sizes below 500 KB. As for the download bandwidth, the error drops greatly in the range above 1000 KB and a number of objects between 200 and 300. These results and observations can give us hints on how to choose web pages to achieve the best possible accuracy.

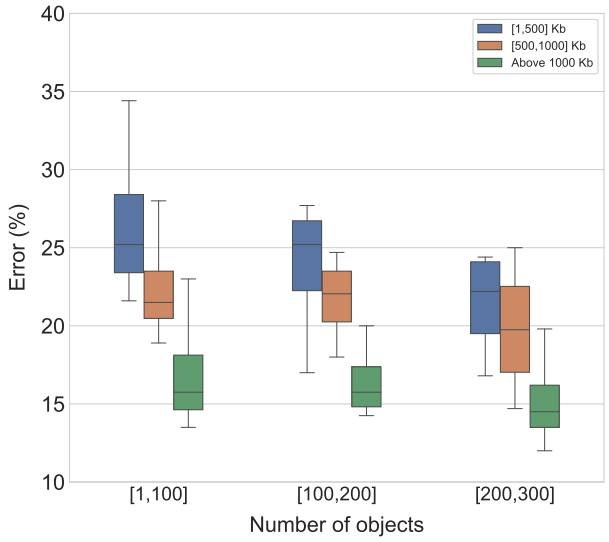
4.3.2 Protocol impact on estimation: HTTP/1.1 vs HTTP/2

We also asked the question if web pages supporting HTTP/2 help to increase the accuracy of estimating delay and download bandwidth, given its novel mixing feature. Since we collect data by browsing real web pages in Google Chrome that require using SSL, we are able to compare the estimation accuracy of both HTTP/2 and HTTP/1.1.

Histograms in Figure 4.7 display, respectively, the estimation error over the same number of web pages with both HTTP/1.1 and HTTP/2 for different ranges of RTT and download bandwidth. While histograms in Figures 4.8 show the RTT and down-



(a) RTT

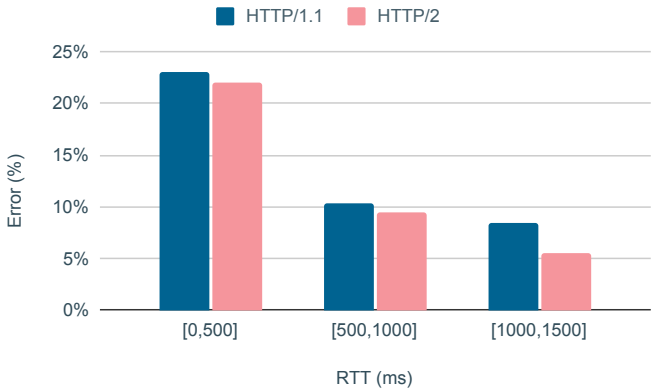


(b) Download bandwidth

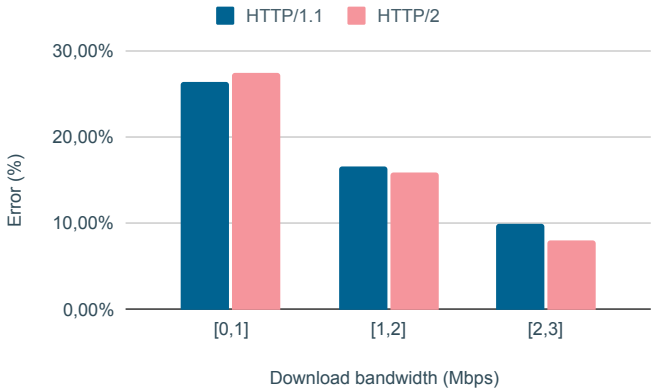
Figure 4.6 – RTT and download bandwidth estimation error in function of the number of objects for different web page sizes

load bandwidth error in function of number of objects for HTTP/1.1 versus HTTP/2.

For RTT, according to Figure 4.7a, the impact on the estimation for HTTP/2 is simi-



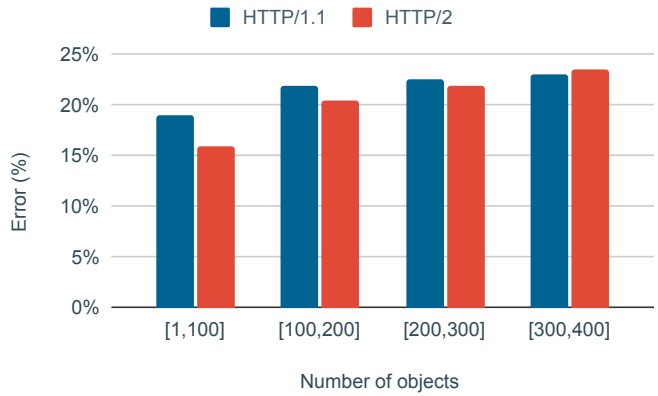
(a) RTT



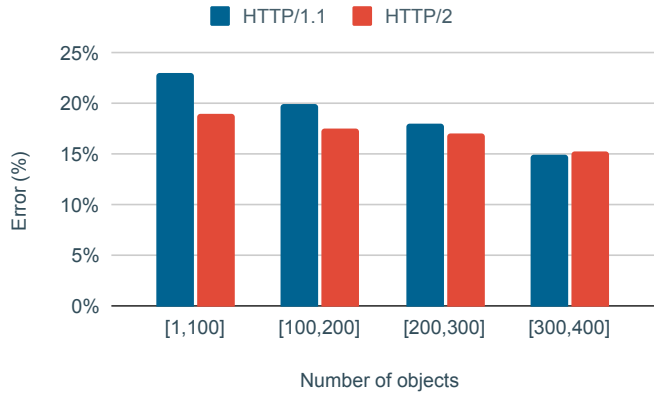
(b) Download bandwidth

Figure 4.7 – Estimation error with both HTTP/1.1 and HTTP/2 for different ranges of delay and bandwidth

lar to HTTP/1.1 for very low RTT. However, for a significantly higher RTT, HTTP/2 give more accurate results than HTTP/1.1. As for the download bandwidth, according to Figure 4.7b, for ranges of bandwidth less than 2 Mbps, the estimation accuracy of HTTP/2 is closely the same as HTTP/1.1. For ranges between 2 Mbps and 3 Mbps HTTP/2 outperforms HTTP/1.1. Further, we notice in Figure 4.8 that considering the number of objects of web pages, HTTP/2 outperforms HTTP/1.1 for ranges between 1 and 200 objects.



(a) RTT



(b) Download bandwidth

Figure 4.8 – RTT and download bandwidth error in function of number of objects for HTTP/1.1 versis HTTP/2

4.4 Our approach against other web-based monitoring solutions

4.4.1 Integrated platform implementation

In this section, we give a detailed overview of the platform used to compare our approach with other web-based monitoring solutions.

We constructed an integrated framework, where we tried to mimic the behavior of the best known troubleshooting tools and services to measure the maximum through-

put and the RTT, by implementing and simulating their behavior. We also integrated in the platform our own approach, which is based on passive measurements in within the browser and deep learning.

As mentioned in Section 2.2.2 about web-based troubleshooting tools, all existing techniques follow a similar approach in measuring latency and bandwidth, which consists of send/receive messages between the web client and the web server. Departing from this observation, we built our experimental framework around an automated process that implements multiple measurement methods including Flash, DOM, XHR, Java Applet, and web socket, see Table 2.1, each of which embedded in a PHP or HTML index page. The Web Client in our platform requests the container web page, renders its elements and executes the measurement codes. This latter step leads to the measurement phase. Finally, the obtained measurements are stored on a dedicated database.

We designed a test bed consisting of a Web Client: Intel Core i7, with 32 GB memory, and a local Web Server squid 3.4.7 where we cached 500 landing pages (see Figure 4.9). In a fully controlled environment, the platform executes the RTT and bandwidth measurements by doing the following: i) set a tuple of (RTT, bandwidth) using a network emulator `tc_config`, ii) check and validating the values using ping tool for latency, and `iperf` tool for bandwidth, iii) start the initialization phase, where the web browser renders the container page, iv) launch the measurements for each scripting technique including our approach, and lastly, (v) store the resulting data.

To reduce measurement bias when calculating baselines in the wild, we repeated the tests for each tuple (RTT, bandwidth) 50 times to retrieve the 500 landing pages and then considered the average values. In order to simulate the internet environment, we introduced an additional delay for each website, consisting of the average of four pings to the real website in normal network conditions.

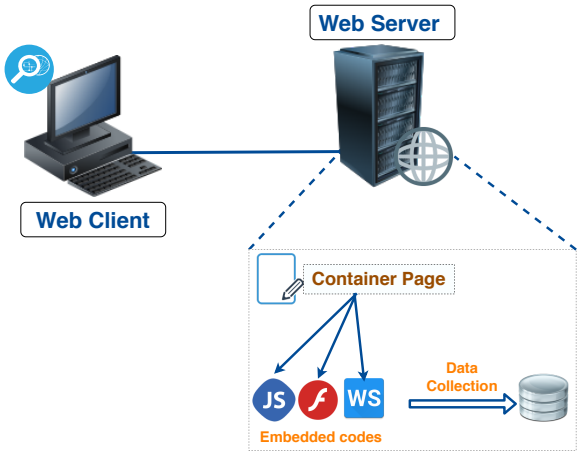


Figure 4.9 – Integrated platform implementation

Table 4.2 – Implemented techniques

	Implemented techniques
HTTP methods	JavaScript: XHR
	JavaScript: DOM
	Flash
Socket methods	Java Applet TCP socket
	Web Socket

4.4.2 Results

Here, we compare the performance of browser-based solutions (see Table 4.2) with our own approach, mainly for measuring two network metrics: RTT and download bandwidth. For each technique, we compare the measured or estimated values with real values. All test conditions were equivalent between our approach and the simulated techniques except for the number of tuples (RTT, bandwidth) generated: our technique is based on passive measurements, hence we need to generate a large dataset if we want to achieve the same accuracy, as active tools. On the other hand, generating the same number for the active tools would be very difficult or even im-

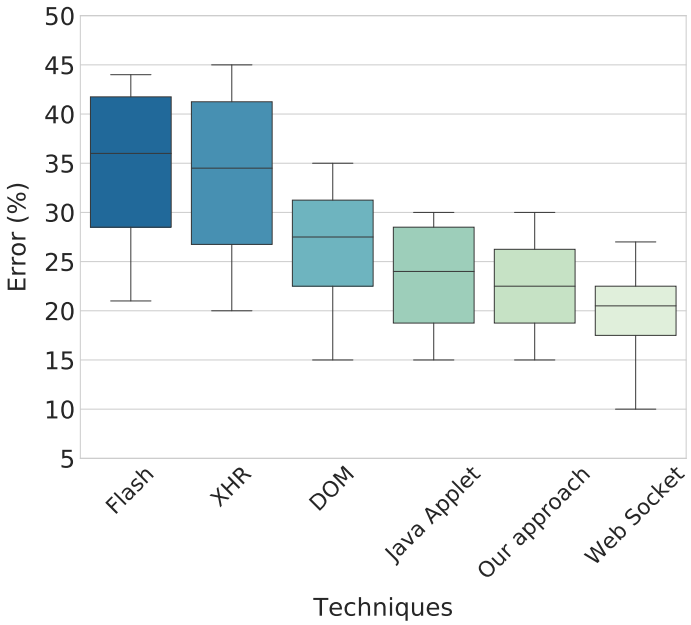


Figure 4.10 – RTT error of implemented troubleshooting techniques

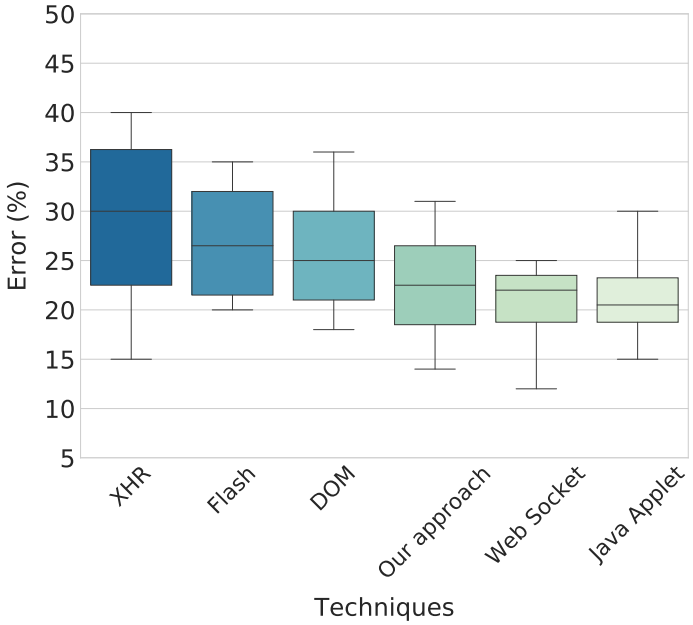


Figure 4.11 – Download bandwidth error of implemented troubleshooting techniques

possible due to the huge amount of time they need to perform measurements. For that purpose, we generated 100 tuples for all the techniques and 5000 tuples for our approach. However, we considered the same ranges for the network metrics: for RTT

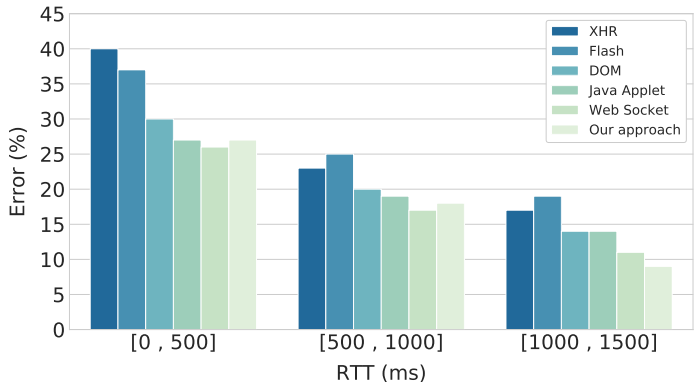


Figure 4.12 – Error of implemented techniques in terms of RTT

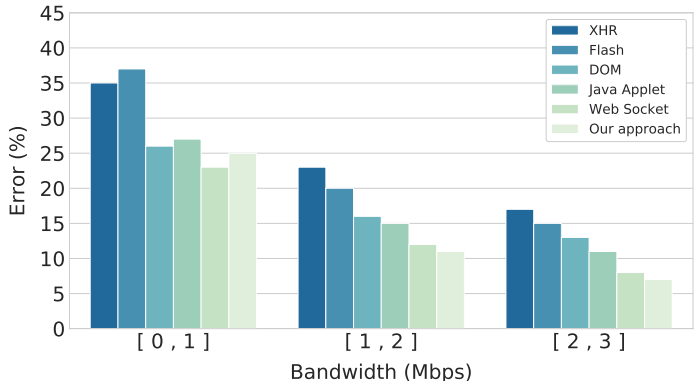
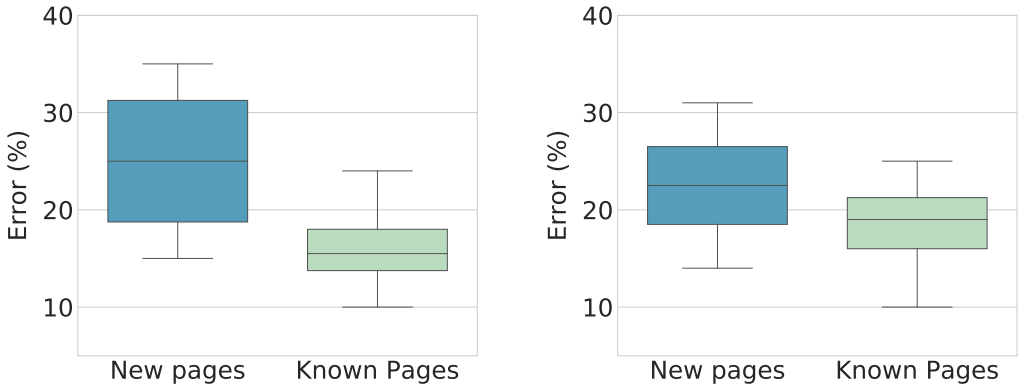
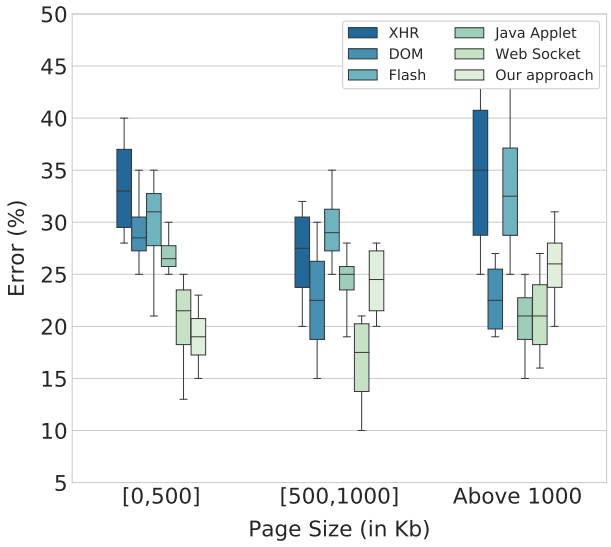


Figure 4.13 – Error of implemented techniques in terms of download bandwidth

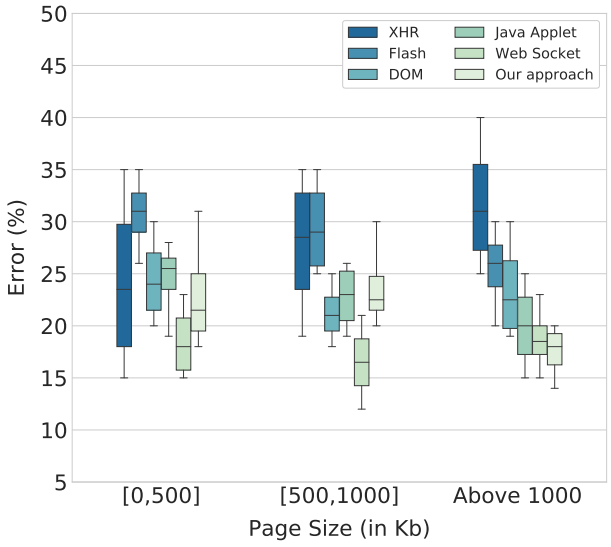


(a) RTT (b) Download bandwidth

Figure 4.14 – Comparison of performance for estimating RTT and bandwidth between pages we know and new pages



(a) RTT

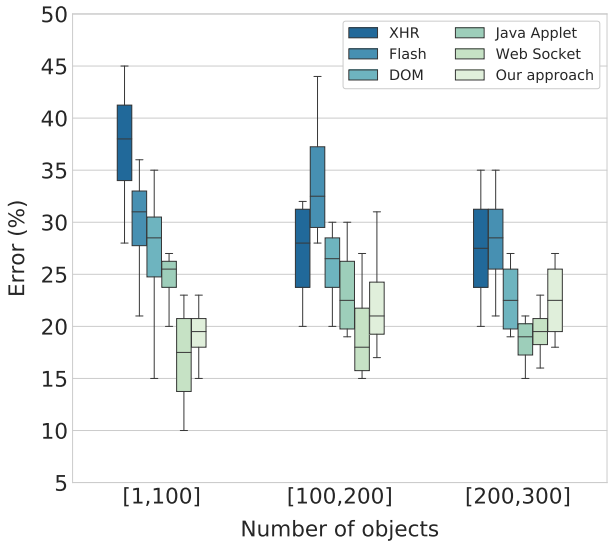


(b) Download bandwidth

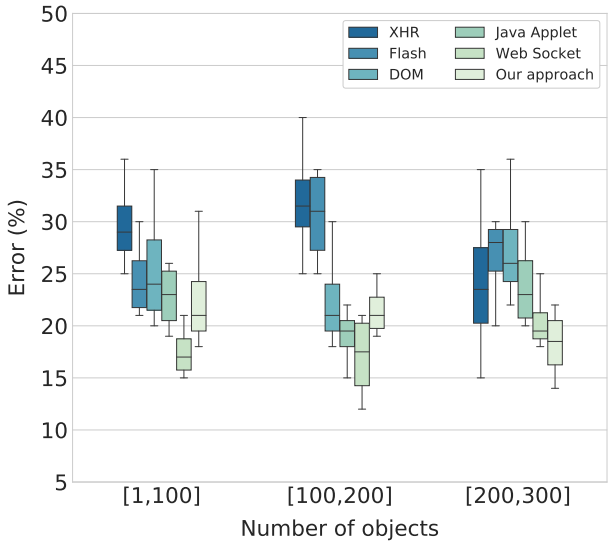
Figure 4.15 – RTT and download bandwidth error in function of page size

it is up to 1500 ms, and for download bandwidth it is up to 3 Mbps.

The box plots in Figures 4.10 and 4.11 display, respectively, the dispersion of error over web pages using the implemented techniques for both metrics: RTT and



(a) RTT



(b) Download bandwidth

Figure 4.16 – RTT and download bandwidth error in function of number of objects

download bandwidth. While histograms in Figures 4.12 and 4.13 compare the implemented solutions to each other in terms of error for different ranges of RTT and download bandwidth. The same as before, the y-axis in the figures shows the mean

percentage error given by Equation (4.2).

$$Error = 100\% \frac{1}{n} \sum_{i=1}^{n} | \frac{Estimated\ or\ Measured\ Value - Real\ Value}{Real\ Value} | \tag{4.2}$$

First, we notice a significant gap between the different techniques. In general, HTTP-based methods including XHR, DOM, and Flash show the least estimation accuracy for both RTT and download bandwidth. For XHR and Flash techniques, the estimation error was extremely high, which can go up to 40%. DOM technique achieves better results than the latter, but the error still cannot be neglected. On the other hand, socket-based solutions are able to provide the best overall accuracy. We can explain this by the fact that HTTP methods add extra headers to retrieve objects from the web server.

Second, our approach based on passive measurements and deep learning outperform most of the implemented solutions, as it ranked the second after the web socket technique for RTT and the third for the download bandwidth, but with a very small difference up to 3% if we consider the mean over web pages. Moreover, it is ranked the first for large values of RTT and download bandwidth (see Figures 4.12, and 4.13). In particular, the error drops to less than 10% for a download bandwidth around 3Mbps.

Another important result concerns the set of web pages used in the estimation. In fact, as shown in Figure 4.14, if we estimate over the set of pages we know during the training phase (different network conditions between training and test though), the accuracy improves significantly for both RTT and download bandwidth.

Now, in order to study the impact of web page characteristics on the accuracy of the different troubleshooting approaches, we plotted box plots displaying the error as function of the page size as well as the number of objects for RTT and download bandwidth metrics. According to Figure 4.15, the performance of our approach increases significantly for certain intervals and outperforms all the other solutions, this occurs when page size is under 500 kb for the RTT (Figure 4.15a), and above 1000

kb for the download bandwidth (Figure 4.15b). The same thing goes for the number of objects (see Figure 4.16).

4.5 Conclusion

In this chapter, we consolidated our approach to infer network performance based on passive measurements freely available from within the web browser and deep learning models to predict RTT and download bandwidth. Indeed, we highlighted the impact of web page characteristics on estimation accuracy. We implemented a methodology consisting mainly of two phases: the data collection phase, where we varied the network conditions and then captured the extended web measurements, and the estimation phase, where we calibrated a Convolutional Neural Network (CNN) to estimate network metrics. Then, we proposed an integrated platform where we implemented our approach as well as several web-based monitoring solutions for comparison purposes. Results of our studies show that our approach can give a very good accuracy compared to others; its accuracy is even higher than most standard techniques and very close to the rest.

In the coming chapter, we will present how we can make this lightweight and accurate monitoring solution to infer performance issues targeting a subset of pages, and detect in real time when anomalies and degraded network conditions occur.

Chapter 5

Leveraging web browsing performance data for network monitoring

5.1 Introduction

In the previous chapter, we illustrated how our approach can infer network performance as precisely as possible, based only on passive measurements easily collected from within the browser seconded by deep learning techniques. Our approach leads to a free, lightweight, and accurate solution that capture the end-to-end path between the web client and the server hosting the website. In this chapter we build upon this work to propose a solution for network troubleshooting and anomaly detection able to provide a very good estimation of the different network conditions of the web traffic and to detect any shift in performance that targets part of or all the user web traffic. In particular, we differentiate between the different network conditions that face the web pages visited by the user, which is essential for network anomaly detection and web browsing troubleshooting. Our main contributions in this chapter can be summarized as follows:

1. We engineer a distributed system that collects measurements at the browser and network levels.
2. We suggest an original network monitoring framework, based on Bayesian Gaus-

sian Mixture Models (BGMM), able to provide information on the underlying network state for the different visited web pages by the user.

3. We propose an algorithm to detect in real-time the occurrence of anomalies and identify web pages affected by them, thus leading to an efficient web browsing troubleshooting solution.

The rest of this chapter is organized as follows. In Section 5.2 we describe in detail our approach for web-based network monitoring using BGMM clustering. In Section 5.3 we present the platform implementation, and then we study the efficiency of our monitoring solution in detecting anomalies. Finally, we conclude in Section 5.4.

5.2 Web-based network monitoring using data clustering

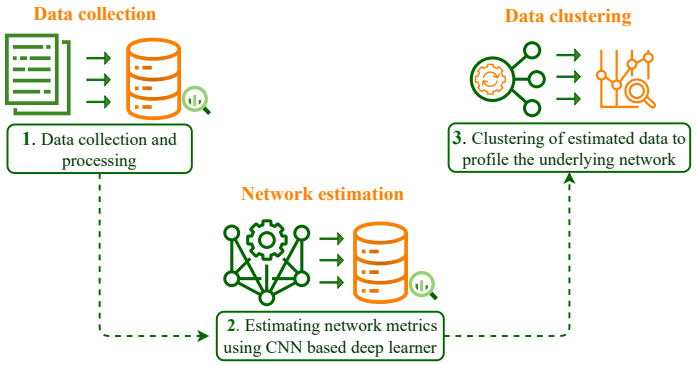


Figure 5.1 – Our appraoch in three phases

Unlike traditional network monitoring methods based on active measurements, our goal is to develop a lightweight solution that deploys a browser-based passive measurement approach. We then use pre-calibrated machine learning models to bridge the gap between the browser-level measurements and the network performance.

In real life, the web browsing traffic generated by the user runs in different network conditions that we don't all know. Knowing the spectrum of these conditions and

tracking them over time is an important step towards understanding the underlying network and detecting the prevalence of anomalies, hitting part or all web browsing activity. For example, an overall change of network performance is a sign of a local network problem. A part of the traffic being degraded is a sign of a remote problem specific to the impacted websites, rather than a general access problem. Given these observations, we propose here a web-based network monitoring framework able to infer underlying network conditions as well as identifying network anomalies for the different visited web pages.

As shown in Figure 5.1, our approach consists of three main phases. First, we collect web measurements from the browser, such as PLT, Connect Start, DNS, and other web page characteristics (see Table 5.1). Next, in order to estimate the network state from the web performance metrics and the page characteristics, we rely on the results of the previous chapter where we propose a Convolutional Neural Network (CNN) model that has as input the different web metrics and as an output the estimation of the RTT (Round-Trip Time) and the download network speed for each of the visited web pages. We then cluster the obtained estimations of network metrics for a set of visited web pages by the user using clustering techniques such as Gaussian Mixture models in order to profile the underlying network. We end up proposing an algorithm to track the clusters in real time to detect any network anomaly and those pages impacted by it.

5.2.1 Data collection

To validate our method, we need labelled network data where the ground truth about the network conditions is known. The purpose of this phase is thus to collect a large dataset that captures the link between web browsing performance and network performance using controlled experiments where we have full control on the network. For this purpose, we have developed a distributed system where different network conditions are emulated and network and web measurements are collected.

Table 5.1 – Web performance metrics

Web QoS metrics	Web page features
Connect Start	Page Size (Size)
DNS	Maximum objects size (Max Objects)
Request	Number of objects (NumObjects)
Response	Protocol supported (Protocol)
DOM	Median of objects size (Median Objects)
First Paint (FP)	Total objects size (Objects' Size)
First Contentful Paint (FCP)	First Quantile of object's size (Q1 Objects)
Page Load Time (PLT)	Third Quantile of objects size (Q3 Objects)
	Maximum objects size (Max Objects)

We use our previous CNN model to estimate the network performance from the web measurements.

To carry out the study, our system involves three main entities; the Experimenter, the Web Client and the Database. The Experimenter generates and sets the network configurations, using a traffic shaping emulation tool, and then launches the Web Client. The Web Client is composed of the browser which is responsible for loading a specific web page, and running an extension that collects all the information that we need to build our model. Finally, the database is where the collected data is stored. We will tackle the framework development thoroughly in the implementation part.

5.2.2 Data-driven network estimation

This phase consists in estimating the underlying network metrics using the collected data from within the browser. Convolutional Neural Networks (CNN) are one of the best estimation techniques capable of automating the process of feature extraction. A CNN is usually followed by a fully connected Neural Network (NN) to guarantee a high prediction accuracy as well as the convergence of the model.

In this section, we use this advanced technique since it shows the highest esti-

mation accuracy compared to widely used traditional machine learning techniques. We consider the regression variant of CNN that we developed earlier in this thesis, and that has as input the different web metrics (Connect Start, DNS, Request, Response, DOM, FP, FCP, PLT), besides some web page related features, such as the number of objects, the size, and the protocol supported. We obtain as output a tuple of estimations $(\hat{rtt}, \hat{bd})$ reflecting the network conditions under which the page was browsed.

5.2.3 Data clustering

Here we give a detailed overview of the Gaussian Mixture Model (GMM) [81] we opt for to perform the clustering of network conditions based on estimations provided by the deep learner. Later on, we justify the efficiency of this clustering algorithm by comparing its performance to other clustering methods.

In general, a mixture model like GMM is a statistical model used to parametrically estimate the distribution of random variables by modelling them as a sum of several other simple distributions. In particular, a Gaussian mixture model is a linear combination of a finite number of Gaussian components with unknown parameters. Assume the existence of a n-dimensional random variable $X = \{x_i | i \in 1, ..., n\}$, the probability density $g(x)$ of the Gaussian's mixture modeling X can be expressed as the weighted sum of M other components whose densities are $g_k(x)$, $k \in 1, ..., M$:

$$g(x, \theta) = \sum_{k=1}^{M} \pi_k g_k(x, \theta_k), \tag{5.1}$$

where π_k represents the prior probability of a data point belonging to component k. The π_k satisfies the probability conditions $\sum_k \pi_k = 1$ and $0 \leq \pi_k \leq 1$. θ and θ_k respectively denote the parameters of the model g and g_k.

We run GMM taking as input the unlabeled estimated values of the underlying network metrics (delay and bandwidth) and as output the predicted clusters of

these metrics together with the parameters of the associated Gaussian components. Several methods exist to estimate these parameters, the widely used one being the Expectation-Maximisation method (EM), which proceeds in an iterative way following the Maximum Likelihood principle.

One of the challenges of the GMM method is how to set the optimal number of clusters; that is, the number of Gaussian components that fit best the data. For that, we use an extension of the EM algorithm built using the Bayesian variational inference technique (BV). For instance, the new method called BGMM [60] for Bayesian Gaussian Mixture Model will eventually not only estimate the cluster parameters but will also give an approximation of the cluster distribution itself. Figure 5.2 displays the flowchart of the BGMM clustering approach.

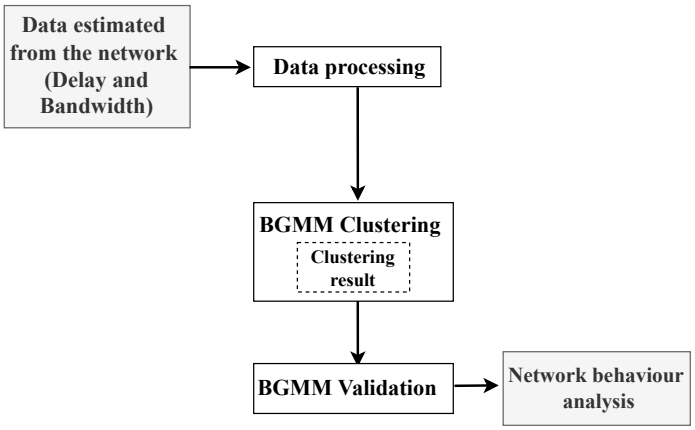


Figure 5.2 – Flowchart for BGMM network monitoring analysis

5.2.4 Clustering validation

Clustering validation is an essential step to assess how good the clustering model is. We consider for this validation three widely used scores: purity (P), Rand index (R) and Fowlkes Mallows Score (FM) [75].

Purity This index measures how pure each cluster is, which means to what extent the elements of a given cluster are included in the ground truth partition. One way

to express the purity of clustering is given by:

$$P = \frac{1}{n} \sum_{i} \max_{j} |C_i \cap T_j|, \tag{5.2}$$

where n is the total number of data points, i is the cluster index and $|C_i \cap T_j|$ is the number of points that are common between the found cluster C_i and the ground truth partition T_j. This purity index is in the range $[0, 1]$; the closer it is to 1, the better the compatibility with the ground truth.

Rand index Given clustering C and ground truth partitioning T, the pairwise measures utilize the partition and cluster label information over all pairs of points. Let (x_i, x_j) be any two different points in T. If both x_i and x_j belong to the same cluster, we call it a positive event, and if they don't belong to the same cluster, we call that a negative event. There are four possibilities to consider, depending on whether there is an agreement between the cluster labels and ground-truth labels. Based on this concept, the rand index is used to express how similar a cluster is to a ground-truth partition and is given by $R = (TP + TN)/N$, where TP denotes the number of true positive events, TN the number of true negatives, and $N = \binom{n}{2}$ the number of pairs in the data set. The rand index has a value between 0 and 1; a higher value indicates a better similarity, which signifies a better clustering performance.

Fowlkes-Mallows index This index measures how well the clustering model performs using two pairwise indicators, pairwise precision and pairwise recall values. Recall measures the number of data points classified correctly over all points in the same ground-truth partition, while precision measures the number of data points classified correctly over all points in the same cluster. The Fowlkes–Mallows index is defined as the geometric mean of precision and recall:

$$FM = \sqrt{recall.precision}. \tag{5.3}$$

The FM index is also between 0 and 1, with 1 being a scenario with perfect clustering, $FP = 0$ & $FN = 0$.

5.2.5 Real-time anomaly detection

Our goal here is to propose a heuristic algorithm with a periodic process of resettlement that allows tracking clusters over time in order to detect network performance anomalies. Let P be the current group of already visited web pages with network performance estimations $(\hat{rtt}, \hat{bd})$ (one pair of values per visited page) and let Q be the group of web pages that are to be visited successively with rate λ. We consider a window of pages W of temporal span T after which we reinitialize the entire process. In this work, T is taken equal to 48 hours. Let $i \in Q$ be a web page that is visited at time t_i within the window, and let B be the baseline distribution of network performance over P. We proceed as follows for each new page i. First, we define $tag(i)$ that checks if the data point i is marginal or not to the baseline distribution, depending on the criterion $p_{\text{value}}(i)$. The marginality of a web page indicates that it is out of the normal conditions, so we mark it in red. If it is not the case, we mark it in blue to indicate that it corresponds to the normal conditions of the network. Second, we apply BGMM based clustering to see if a new network state emerges. A perfect scheme will be performing clustering for each point i. However, one must consider reducing the cost of calculation of our clustering model. So we introduce a step K (K=10) of re-clustering (only marginal pages are considered in the count). Third, by checking the clustering results, we have two possibilities; either there is an emergence of a new cluster or not. A new cluster composed of red points is a clear sign of a change in network conditions. If the change is to the worst, we return anomaly detected and the pages impacted by the anomaly.

The algorithm resettlement to initialize the whole process considers two cases:

— If there is an emergence of a new network condition, we change the baseline distribution in order to adapt it to the new network state.

— If no emergence of a new state occurs, we wait until T time units elapse, then we reset the group P as well as its baseline distribution B.

5.2.6 Anomaly detection analysis

For a better understanding of the anomaly, we consider the following elements:

— *Anomaly type and magnitude:* which means having either an increasing delay or a bandwidth drop and by how much compared to the precedent network state.

— *Probable source of the anomaly:* which consists in identifying which part of the network bears the major responsibility for the anomaly; the local part or the WAN part.

— *Anomaly duration:* the time between the anomaly is detected, and the network conditions move back to normal.

Now, let E_m be the cluster that represents the detected anomaly, and P be the set of visited web pages. If web pages of E_m are only a partial subset of P, we say that P changed partially (change of network conditions for the same pages). In this case, the anomaly source will very likely depend on the other part of the Internet. On the other hand, if all web pages of P are in the emerged cluster, we say that P changed totally, and the source of degradation is local (the access network).

5.3 Performance evaluation

5.3.1 Framework setup

We built an experimentation framework that integrates our approach based on the joint use of passive measurements and deep learning (CNN). Our framework is built around an automated process that consists of sending and receiving messages between the Experimenter and the Web Client. The Experimenter is divided into a simple Finite State Machine (FSM), developed in python, and a web page to load the

experiments. For the Web Client, we use Google Chrome as a browser, and interact with it via a JavaScript extension we developed for the purpose of the study, using Chrome Navigation Timing API and Performance Navigation API, which are w3c recommendations (see Figure 5.3).

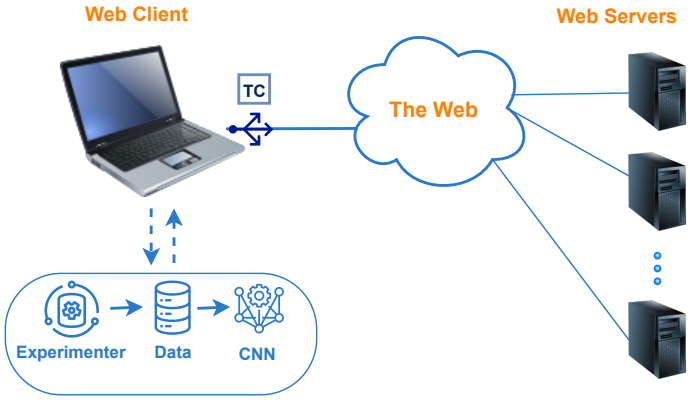


Figure 5.3 – Framework implementation

We define the network conditions we want to emulate as N-tuples defined as follows: $\{TC_1^i, ..., TC_N^i\}$ where N (for example $N \in \{2, ..., 5\}$) is the number of different network conditions faced by the user at a random time, i is the i-th experimentation and $TC = (RTT, BD, p)$ is a tuple of network delay RTT, network bandwidth BD and p being a group of web pages visited by the user and assigned to a particular network condition (RTT and BD). The values of RTT and bandwidth are picked randomly from a list of samples generated by the FAST technique (Fourier Amplitude Sensitivity Test [88]); FAST is a sampling method that covers a given space based on relevant frequencies, which allows an efficient scan of the area to be sampled. These values are then enforced by the network emulator using Linux Traffic Control tool *tc-config*. The group of web pages p is chosen by picking randomly $|p|$ web pages from the total group of web pages P, such that: $|p| = \lceil \frac{|P|}{N} \rceil$. In this work, we consider as P the 500 top popular web pages according to Alexa ranking.

Having under our control the end to end network path can be a real challenge; in fact, we have to load real web pages from the cloud. Unfortunately, this latter is out

of our experimental network. In such circumstances, *tcconfig* may face difficulties in enforcing the desired configurations. Thus, the need to validate and check samples of the N-tuples (RTT and bandwidth) before starting the actual experimentation, through RTT noise estimation and throughput tests.

Our platform performs the measurements of web performance metrics listed in Table 5.1, then gives estimations of delay and bandwidth for each visited page using the deep learner CNN we proposed in Chapter 3 [87, 86]. By applying our approach (all scenarios), we obtain a dataset composed of 8000 different network scenarios for the 500 considered web pages.

5.3.2 BGMM tuning

GMM possesses several hyper-parameters to be tuned. The most important one is the number of mixture components or clusters, `n_components`.

The Bayesian Gaussian Mixture Model (BGMM) determines the optimal number of clusters by affecting some clusters `weights` to zero, depending on the value of the criterion `weight concentration prior`, so the number of clusters found is always smaller than `n_components`. The `covariance_type` describes the type of covariance parameters to use. And `weight_concentration_prior` describes the type of the weight concentration prior, either using the Stick-Breaking representation (Dirichlet process) or by favoring more uniform weights using Dirichlet distribution. We also need to choose `init_params`, which decides about the method used to initialize the weights, the means, and the covariances.

For our study, we consider `n_components` = 5, since in our experiments, we only consider scenarios with maximum possible five TC configurations. For the `covariance_type`, we use 'full' since we want to find completely separated clusters with singular covariance matrix each. Regarding the `weight_concentration_prior`, we use Dirichlet process, and finally we set 'k-means' as initialization method in `init_params`.

5.3.3 Results

We run BGMM over the tuples of estimated bandwidth and RTT. We tune BGMM to find automatically the optimal number of clusters, in the limit of 5 clusters.

General case The box plot in Figure 5.4 displays the dispersion of the global clustering accuracy over all network configurations. The histogram in Figure 5.5 gives the accuracy of finding the same number of clusters as the ground truth. In both figures, the x-axis represents the number of different network configurations which are supported by the experimentation. The y-axis shows the accuracy in percentage defined using the Rand index R defined above.

We notice a minor gap between the different scenarios, with the accuracy decreasing with the number of parallel network conditions. In general, the 5 TCs scenario shows the least accuracy for both the clustering and the identification of the right number of clusters. Still, the value is as high as 85% for the first metric and 90% for the second metric. For the 2 TCs scenario, the accuracy of clustering is even higher (90% on average), and the precision of finding the exact number of clusters is perfect (near 100%). As for the 3 TCs and 4 TCs scenarios, they display an accuracy higher than 85% for the clustering as a whole and 95% for the simpler problem of finding the right number of clusters.

Now, we check whether the accuracy of clustering varies if we change the number of web pages until we reach the maximum 500. We consider all the scenarios in Figure 5.6 and show a CDF plot displaying the Accuracy as a function of the number of visited web pages. We can notice how the performance of our approach increases significantly with the number of web pages. For example, to reach an accuracy of 75% we need 50 web pages for 2 TCs, 175 web pages for 3 TCs, 225 for 4 TCs, and 500 for 5 TCs. These results give hints on how to choose the minimum number of web pages to consider for clustering to achieve the best possible accuracy.

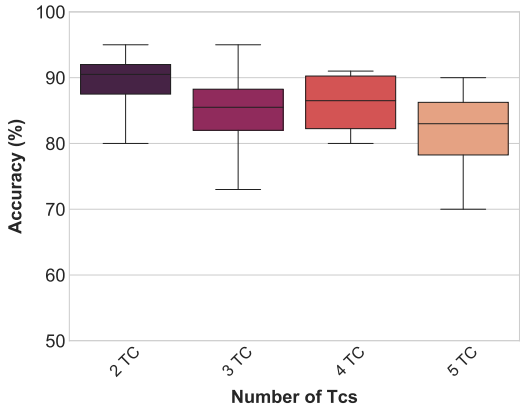


Figure 5.4 – BGMM clustering accuracy for 500 web pages

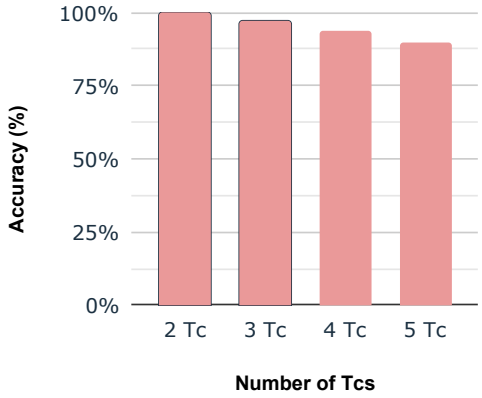


Figure 5.5 – Accuracy of finding the right number of clusters

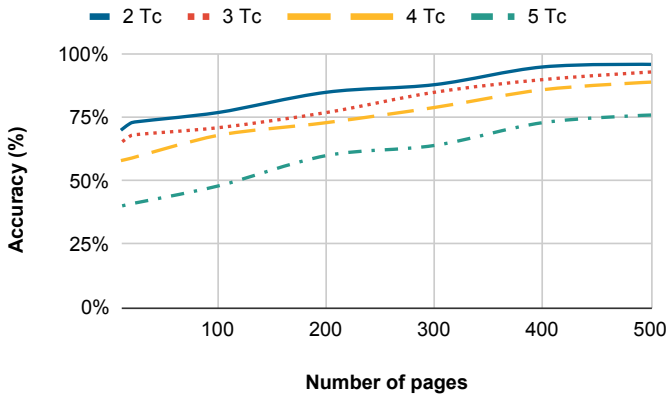


Figure 5.6 – Clustering accuracy versus number of pages

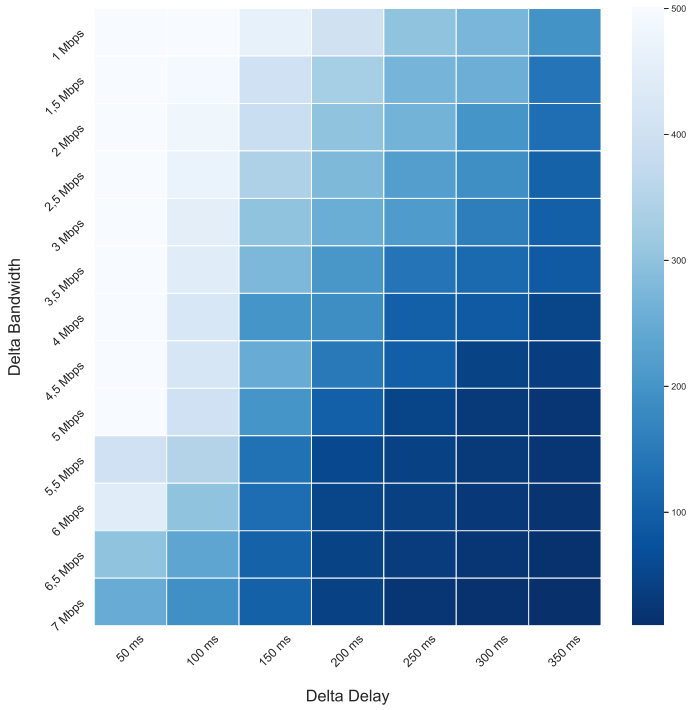


Figure 5.7 – Heat map of the minimum number of pages needed to achieve 85% clustering accuracy

Case of two TCs The identification of the different network conditions depends on the distance that exists between these conditions. We study this relationship for the case of two different conditions (N=2). We plot in Figure 5.7 the heatmap of the minimum number of pages needed to achieve 85% of accuracy, for different distances in terms of delay and bandwidth. Clearly, the more distant the conditions, the fewer the number of pages required, with a few pages being sufficient to separate conditions differing by 7Mbps and 300ms, and 500 pages being needed to differentiate scenarios with differences of less than 1Mbps and 50ms.

We further provide results where the number of pages is unbalanced between the two conditions. For some specific scenarios, Figure 5.8 shows that the balance helps in improving clustering accuracy, and any unbalance can be compensated by increasing the number of pages to cluster.

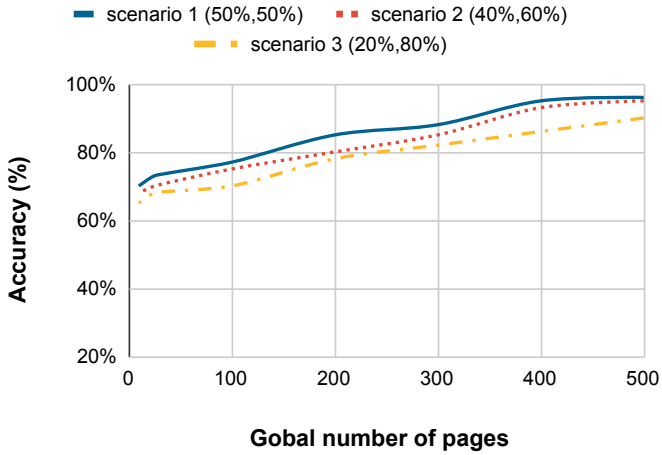


Figure 5.8 – Clustering accuracy versus number of pages for different balances between the network conditions

5.3.4 Comparison with other clustering methods

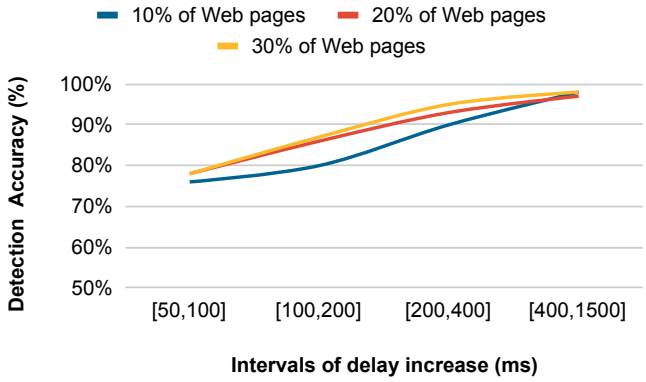
Table 5.2 – Comparison between clustering models

	Purity Index	Rand Index	FM score
K-means	0.502	0.662	0.67
GMM	0.84	0.772	0.78
BGMM	0.89	0.886	0.812

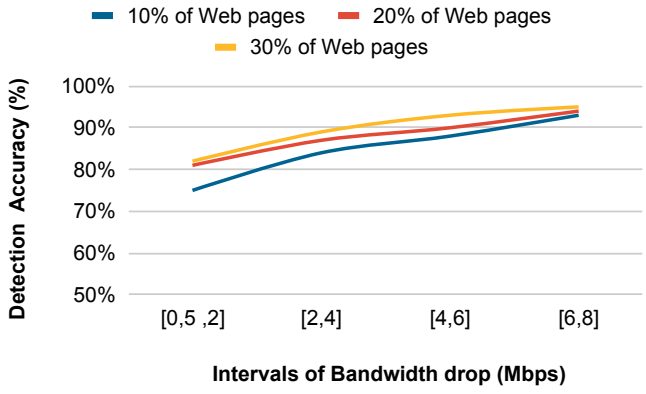
Here, we compare the performance of our BGMM based approach with other two well known clustering methods: K-means [80] and classical GMM [81]. For K-means, we use a model with N components fit. For GMM, we use a model with N components fit and Expectation-Maximization for parameters' estimation. We consider the Elbow Method to determine this optimal number of clusters (between 2 and 5).

As shown in Table 5.2, K-means shows the worst clustering performance, especially for the purity index, which illustrates the difficulty of the task and the need for sophisticated models. GMM comes next, then BGMM; in particular, the purity

index can rise up to 0.89 overall with BGMM.



(a) Increase of delay



(b) Drop of Bandwidth

Figure 5.9 – Accuracy detection of delay increase and Bandwidth drop for different percent of web pages

5.3.5 Anomaly detection validation

Here we validate our algorithm for anomaly detection. We consider two different scenarios with specific anomalies. The first one is a drop in the bandwidth value for a percentage of web pages (10%, 20% or 30%). We variate the drop from 0.5 Mbps to 8 Mbps. The second one consists of an increase in the delay respectively for 10%, 20% and 30% of web pages.

We validate our algorithm over 100 visited web pages (maximum anomaly dura-

tion). Indeed, we randomly pick 200 web pages with homogeneous network conditions from the dataset. We consider 100 pages as the group P representing the baseline distribution. We consider the rest as the group Q representing the web pages that arrive after the anomaly occurs. We include the anomaly in the group Q for a percentage of the pages, then launch the algorithm.

We want to check the ability of our algorithm to detect these anomalies. Figure 5.9 gives an overview of the results. It shows the detection accuracy of the anomalies depending on whether we are dealing with a drop in bandwidth or an increase in RTT. We observe that the accuracy in detecting anomalies increases significantly when we have larger shifts in network performance. Anomaly detection accuracy also increases when the number of pages impacted by the anomaly increases. For example, if the drop in bandwidth is between 6 Mbps and 8 Mbps, the accuracy of detection can go up to 93% for all considered scenarios. An increase in delay between 400 ms and 1500 ms gives an accuracy that can go up to 98%. In all our scenarios, the accuracy was found to be above 75%. Note that one can further increase the accuracy if the anomaly lasts longer than the 100 pages limit we consider here.

5.4 Conclusion

We presented and implemented a lightweight web-based network monitoring solution able to infer the underlying network performance and detect network anomalies. Our solution consists of:

— An original network monitoring framework, based on Bayesian Gaussian Mixture Models (BGMM), able to provide information on the underlying network state for the different visited web pages by the user.

— An algorithm to detect in real time the occurrence of performance anomalies.

We validated our approach with controlled experiments where the network conditions were varied while the web pages were browsed. Validation results showed that

our approach can yield accurate detection results even in scenarios with small shifts in network performance and a few percentage of web pages impacted. We will keep developing this solution towards a deployment in the wild and the identification of the root causes of the degraded network performance based on the identified subset of pages impacted by the anomaly.

Chapter 6

Conclusion and perspectives

6.1 Conclusion

In this thesis, we were mainly interested in inferring the main properties of the underlying network, assessing the network quality and detecting performance anomalies. In a nutshell, we presented new approaches and frameworks to derive our inference models.

First, we presented in Chapter 3 an efficient and novel method to infer network performance metrics from web metrics that can be collected passively and easily from within the browser. For that purpose, We developed a platform to collect a large dataset that links the web performance measurements to the underlying network conditions over an example of three network metrics, namely the round-trip time (RTT), the download bandwidth and the loss rate. After that, we resorted to machine learning to calibrate our estimation models. Also, by comparing deep learning algorithms to classical ML algorithms such as Random Forest, we highlighted the need for sophisticated deep learning algorithms, namely convolutional neural networks (CNN). Results show that our solution gives a very good accuracy by only using measurements passively obtained from within the browser, especially with the CNN model that outperforms the traditional Machine learning (ML) techniques.

Then, in Chapter 4, we highlighted the impact of web page characteristics and protocol supported on the estimation accuracy of our models. To that aim, we proposed a methodology consisting mainly of two phases: the data collection phase, where we

varied the network conditions and then captured specific web measurements with information on the characteristics of the visited web pages (e.g., web page size, number of objects, Protocol supported, etc.)., and the estimation phase, where we calibrated a Convolutional Neural Network (CNN) to estimate network metrics, precisely RTT and Download bandwidth. Moreover, we studied the efficiency of our approach by comparing it experimentally with other network troubleshooting solutions using an integrated platform where we implemented our methodology as well as several web-based monitoring solutions. The accuracy of our approach was outstanding compared to others; indeed, it is higher than most standard techniques and very close to the rest.

Finally, in Chapter 5, we presented and implemented an original network monitoring framework based on Bayesian Gaussian Mixture Models (BGMM), able to provide information on the underlying network state for the different visited Web pages by the user. Furthermore, we proposed an algorithm to detect in real time the occurrence of performance anomalies. Validation results showed that our approach could yield accurate detection even in scenarios with small network performance shifts and a few percentages of web pages impacted. All these contributions lead to an efficient, lightweight, and data-driven network monitoring and troubleshooting web-based solution that runs as a plugin in the end users' browser.

6.2 Perspectives

6.2.1 Extension to further contexts

In this thesis, we leveraged passive measurements freely available in the browser and deep learning techniques to infer network performance without adding new measurement overhead. In Chapter 3 we proposed and evaluated a framework to infer the main properties of the underlying network (the delay, the bandwidth, and the loss rate) with a particular focus on web services. However, our approach remains

applicable to further contexts when it comes to network performance inference. For example, we can extend this work to cover other internet services, such as video streaming, which is one of the principal contributors to global internet traffic. To that aim, we need to understand the transmission process of this service (e.g., dynamic adaptive streaming over HTTP (DASH) protocol) and identify new metrics correlated to video quality and related to video play-out, including the video startup delay, video interruptions or stalls, and resolution switches, etc.

Another perspective is to extend and validate our work on other technologies, particularly wireless communications (e.g., WiFi). Thus, applying our approach relies on adding metrics specific to WiFi networks, such as link-layer metrics related to frame losses due to collision or low signal and the received signal strength indicator (RSSI) that measures the signal strength. This study will allow us to explore an essential direction for future work: test the same approach on mobile phones and tablets, since all our experiments were conducted on desktops using a Chrome plugin. Today, smartphones and tablets are becoming more powerful; according to Statista, 53.62% of all website traffic was due to mobile phones in 2020. Incorporating our models in a new mobile application that measures web performance and predicts the network state from mobile web data will be a good starting point. However, performing traffic collection and automatic users' network quality assessment is a challenging task.

Moreover, our methodology can be developed to infer more network properties, for example, network neutrality. This concept implies that data on the Internet should be treated the same way regardless of the service provider, technology, or country they come from. Accordingly, we examine if our approach can identify whether web pages are handled differently or they are treated the same.

6.2.2 Crowdsourcing and Federated Learning (FL)

As highlighted in Chapter 3, Our deep learning models are calibrated using a controlled experimentation approach where we artificially change the network conditions and automate a web browsing activity. Then we use the ground truth on the network state and the passive measurements available in the browser for the training and validation of our models. The particularity of our solution is that it covers all the possible network scenarios, especially extreme ones, where for example, latencies are too high, or bandwidth is too large. However, to create more robust and precise models that fit real life, we would like to leverage and incorporate real scenarios where network traffic is dynamic and where the network state is not necessarily known beforehand. So one future direction is to consider data collected from the wild through crowdsourcing.

Our idea consists of deploying a Google Chrome plugin that measures web and network metrics using passive/active techniques. Therefore, we will achieve a scalable and efficient solution able to reach a large group of real users and collect much more data in a short time. Then we use the data collected to re-calibrate our machine learning models. Once these models are ready, network performance can thus be estimated from web performance measurements without the need to access or probe the network.

Accordingly, this work can be extended to another interesting aspect based on federated learning (FL), a newly proposed machine learning method that uses a decentralized dataset. In federated learning, edge devices collaboratively build a unified learning model by sharing only locally learned models while keeping the local training data.

We imagine developing a system able to work in a distributed manner by sharing individual models specific to each end user and producing one more accurate and precise model by implementing FL algorithms. Such an approach has many advantages. First, it allows increasing privacy since only models are shared. Second, it helps to

improve learning performance and achieve higher accuracy. Third, it consumes fewer resources since it doesn't require exchanging voluminous training data.

6.2.3 Localization of anomalies

In Chapter 5, we presented and implemented a lightweight web-based network monitoring solution to infer the underlying network performance and detect abnormal network behaviors. The validation of our framework showed that our solution could produce accurate detection results. However, to better understand the anomaly, a complete study on the localization of anomalies is needed.

To that aim, we think of keeping developing this solution towards identifying the root causes of the degraded network performance based on the identified subset of pages impacted by the anomaly. In particular, finding which part of the network bears the major responsibility for the anomaly is a fundamental starting block. Is it from the client device, the local access, the remaining cloud or the server-side?

We will opt for a solution that, besides deducing any significant change in real time, is able to analyze and understand its root causes. Indeed, to localize anomalous events, we will envisage access to further network-level traces as the BGP routing tables and the traceroutes. Also, we can investigate refined strategies that consider a more realistic network topology.

PUBLICATIONS

Conference Papers

- Imane Taibi, Yassine Hadjadj-Aoul, and Chadi Barakat: **When Deep Learning meets Web Measurements to infer Network Performance** _ *CCNC 2020-IEEE Consumer Communications & Networking Conference, Las Vegas, United States*: IEEE, Jan. 2020, pp. 1–6.

- Imane Taibi, Yassine Hadjadj-Aoul, and Chadi Barakat: **Data Driven Network Performance Inference From Within The Browser**_ *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1-6.

- Imane Taibi, Yassine Hadjadj-Aoul, and Chadi Barakat: **Leveraging Web browsing performance data for Network monitoring: A data-driven approach**_ *2022 IEEE Global Communications Conference: Communication QoS, Reliability and Modeling - Communication QoS, Reliability and Modeling.*

BIBLIOGRAPHY

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu, « A survey of network anomaly detection techniques », *in*: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31, ISSN: 1084-8045, DOI: https://doi.org/10.1016/j.jnca.2015.11.016.

[2] Vaibhav Bajpai and Jürgen Schönwälder, « A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts », *in*: *IEEE Communications Surveys &amp Tutorials* 17 (Apr. 2015), DOI: 10.1109/COMST.2015.2418435.

[3] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang, « Developing a Predictive Model of Quality of Experience for Internet Video », *in*: *SIGCOMM Comput. Commun. Rev.* 43.*4* (Aug. 2013), pp. 339–350, ISSN: 0146-4833.

[4] Othmane Belmoukadam and Chadi Barakat, « Unveiling the end-user viewport resolution from encrypted video traces », *in*: *IEEE Transactions on Network and Service Management* 18.*3* (Sept. 2021), pp. 3324–3335, DOI: 10.1109/TNSM.2021.3083070, URL: https://hal.inria.fr/hal-03230168.

[5] Othmane Belmoukadam, Thierry Spetebroot, and Chadi Barakat, « ACQUA: A user friendly platform for lightweight network monitoring and QoE forecasting », *in*: *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, 2019, pp. 88–93.

[6] Mike Belshe, Roberto Peon, and Martin Thomson, *Hypertext transfer protocol version 2 (HTTP/2)*, 2015.

[7] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk, *Hypertext transfer protocol–HTTP/1.0*, 1996.

[8] Timothy J Berners-Lee and Robert Cailliau, « WorldWideWeb: Proposal for a HyperText project », *in*: (1990).

[9] M Bishop and E Akamai, « Hypertext transfer protocol version 3 (HTTP/3) draft-ietf-quic-http-18 », *in*: *Internet Requests for Comments, IETF Internet Draft, Tech. Rep.* (2019).

[10] 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*, ed. by Mohamed Ali Kaafar, Steve Uhlig, and Johanna Amann, Cham: Springer Int. Publishing, 2017, pp. 47–59.

[11] Enrico Bocchi, Luca De Cicco, and Dario Rossi, « Measuring the Quality of Experience of Web users », *in*: *ACM SIGCOMM Computer Communication Review* 46 (Dec. 2016), pp. 8–13, DOI: 10.1145/3027947.3027949.

[12] Tim Bray, « Measuring the Web », *in*: *World Wide Web J.* 1.*3* (1996).

[13] Kjell Brunnström, Sergio Ariel Beker, Katrien De Moor, Ann Dooms, Sebastian Egger, Marie-Neige Garcia, Tobias Hossfeld, Satu Jumisko-Pyykkö, Christian Keimel, Mohamed-Chaker Larabi, Bob Lawlor, Patrick Le Callet, Sebastian Möller, Fernando Pereira, Manuela Pereira, Andrew Perkis, Jesenka Pibernik, Antonio Pinheiro, Alexander Raake, Peter Reichl, Ulrich Reiter, Raimund Schatz, Peter Schelkens, Lea Skorin-Kapov, Dominik Strohmeier, Christian Timmerer, Martin Varela, Ina Wechsung, Junyong You, and Andrej Zgank, *Qualinet White Paper on Definitions of Quality of Experience*, Mar. 2013, URL: https://hal.archives-ouvertes.fr/hal-00977812.

[14] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, « Characterizing Web Page Complexity and Its Impact », *in*: *IEEE/ACM Transactions on Networking* 22.*3* (June 2014), pp. 943–956.

[15] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar, « Characterizing Web Page Complexity and Its Impact », *in*: *IEEE/ACM Transactions on Networking* 22.*3* (2014), pp. 943–956, DOI: 10.1109/TNET.2013.2269999.

[16] Tom Callahan, Mark Allman, and Vern Paxson, « A longitudinal view of http traffic », *in*: *International Conference on Passive and Active Network Measurement*, Springer, 2010, pp. 222–231.

[17] CERN, *The birth of the Web*, https://home.cern/science/computing/birth-web.

[18] Kuan-Ta Chen, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei, « A Crowdsourceable QoE Evaluation Framework for Multimedia Content », *in*: MM '09, Beijing, China: Association for Computing Machinery, 2009, pp. 491–500, ISBN: 9781605586083, DOI: 10.1145/1631272.1631339.

[19] Heng Cui, Ernst Biersack, and Eurecom Sophia Antipolis, *Distributed Troubleshooting of Web Sessions Using Clustering*.

[20] Diego 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*: *PAM 2018 - Int. Conf on Passive and Active Network Measurement*, Berlin, Germany, Mar. 2018, pp. 1–13.

[21] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu, « Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems », *in*: *Proceedings of the 9th international conference on Autonomic computing*, 2012, pp. 191–200.

[22] Carlo Demichelis and Philip Chimento, *IP packet delay variation metric for IP performance metrics (IPPM)*, tech. rep., 2002.

[23] Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson, « Fathom: A Browser-based Net-

work Measurement Platform », *in*: *ACM Internet Measurement Conference*, Boston, United States: ACM, Nov. 2012, pp. 73–86.

[24] DOM, https://dom.spec.whatwg.org/.

[25] KIT EATON, *How One Second Could Cost Amazon $1.6 Billion In Sales*, https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales, 2012.

[26] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K Ramakrishnan, « Towards a spdy'ier mobile web? », *in*: *IEEE/ACM Transactions on Networking* 23.*6* (2015), pp. 2010–2023.

[27] Rodérick Fanou, Gareth Tyson, Pierre Francois, and Arjuna Sathiaseelan, « Pushing the Frontier: Exploring the African Web Ecosystem », *in*: *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 435–445, ISBN: 9781450341431, DOI: 10.1145/2872427.2882997.

[28] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis, « The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic », *in*: *Proceedings of the ACM Internet Measurement Conference*, IMC '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1–18, ISBN: 9781450381383, DOI: 10.1145/3419394.3423658, URL: https://doi.org/10.1145/3419394.3423658.

[29] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L Wiener, « A large-scale study of the evolution of Web pages », *in*: *Computing Reviews* 46.*1* (2005), p. 51.

[30] M. Fiedler, T. Hossfeld, and P. Tran-Gia, « A generic quantitative relationship between quality of experience and quality of service », *in*: *IEEE Network* 24.*2* (Mar. 2010), pp. 36–41.

[31] Roy T. Fielding and Julian Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, RFC 7231, June 2014, DOI: 10.17487/RFC7231, URL: https://www.rfc-editor.org/info/rfc7231.

[32] Andrew Froehlich, *DEFINITION of bandwidth (network bandwidth)*, https://www.techtarget.com/searchnetworking/definition/bandwidth.

[33] Song Fu, Jianguo Liu, and Husanbir Singh Pannu, « A Hybrid Anomaly Detection Framework in Cloud Computing Using One-Class and Two-Class Support Vector Machines », *in*: *ADMA*, 2012.

[34] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad, « Perceived Performance of Top Retail Webpages In the Wild: Insights from Large-scale Crowdsourcing of Above-the-Fold QoE », *in*: *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, Internet QoE '17, Los Angeles, CA, USA: ACM, 2017, pp. 13–18.

[35] Xiaohui Gu and Yongmin Tan, « Online performance anomaly prediction and prevention for complex distributed systems », *in*: 2012.

[36] Qiang Guan, Ziming Zhang, and Song Fu, « Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems. », *in*: *J. Commun.* 7.*1* (2012), pp. 52–61.

[37] Andreas Hanemann, Athanassios Liakopoulos, Maurizio Molina, and Martin Swany, « A study on network performance metrics and their composition », *in*: *Campus-Wide Information Systems* 23 (Aug. 2006), DOI: 10.1108/10650740610704135.

[38] Félix Hernández-Campos, Kevin Jeffay, and F Donelson Smith, « Tracking the evolution of web traffic: 1995-2003 », *in*: *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* IEEE, 2003, pp. 16–25.

[39] 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*: *International Conference on Passive and Active Network Measurement*, Springer, 2018, pp. 31–43.

[40] Tobias Hoßfeld, Florian Metzger, and Dario Rossi, « Speed index: Relating the industrial standard for user perceived web performance to web qoe », *in*: *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, 2018, pp. 1–6.

[41] Tobias Hossfeld, Poul Heegaard, Martín Varela, and Sebastian Möller, « QoE beyond the MOS: an in-depth look at QoE via better metrics and their relation to MOS », *in*: *Quality and User Experience* 1 (Sept. 2016), p. 2, DOI: 10.1007/s41233-016-0002-1.

[42] *HTML basics*, https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.

[43] Evolution of HTTP, https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP.

[44] Tian Huang, Yan Zhu, Qiannan Zhang, Yongxin Zhu, Dongyang Wang, Meikang Qiu, and Lei Liu, « An lof-based adaptive anomaly detection scheme for cloud computing », *in*: *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, IEEE, 2013, pp. 206–211.

[45] *Identifying resources on the Web*, https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web.

[46] *IP Performance Measurement*, https://datatracker.ietf.org/wg/ippm/documents/.

[47] Manish Jain and Constantine Dovrolis, « Pathload: A Measurement Tool for End-to-End Available Bandwidth », *in*: *Proc. of Passive and Active Measurement Workshop* (Mar. 2002).

[48] Artur Janc, Craig E. Wills, and Mark Claypool, « NETWORK PERFORMANCE EVALUATION IN A WEB BROWSER », *in*: proceeding of IASTED PDCS, 2009.

[49] Michalis Katsarakis, Renata Cruz Teixeira, Maria Papadopouli, and Vassilis Christophides, « Towards a Causal Analysis of Video QoE from Network and Application QoS », *in*: *Proc. of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, Internet-QoE '16, Florianopolis, Brazil: ACM, 2016, pp. 31–36, ISBN: 978-1-4503-4425-8.

[50] Muhammad Jawad Khokhar, Thibaut Ehlinger, and Chadi Barakat, « From Network Traffic Measurements to QoE for Internet Video », *in*: *IFIP Networking Conference 2019*, Varsovie, Poland, May 2019, DOI: 10.23919/IFIPNetworking.2019.8816854, URL: https://hal.inria.fr/hal-02074570.

[51] Muhammad Jawad Khokhar, Nawfal Abbassi Saber, Thierry Spetebroot, and Chadi Barakat, « An intelligent sampling framework for controlled experimentation and QoE modeling », *in*: *Computer Networks* 147 (2018), pp. 246–261, ISSN: 1389-1286.

[52] Diederik P. Kingma and Jimmy Ba, *Adam: A Method for Stochastic Optimization*, 2014, arXiv: 1412.6980 [cs.LG].

[53] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson, « Netalyzr: Illuminating the edge network », *in*: Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, Jan. 2010, pp. 246–259, DOI: 10.1145/1879141.1879173.

[54] Pierre L'Ecuyer, « Randomized Quasi-Monte Carlo: An Introduction for Practitioners », *in*: *12th Int. Conf. on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing (MCQMC 2016)*, Stanford, United States, June 2017.

[55] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan 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*: *Proc. of the Conf. of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, Los Angeles, CA, USA: ACM, 2017, pp. 183–196.

[56] Jong-Seok Lee, Francesca De Simone, Naeem Ramzan, Zhijie Zhao, Engin Kurutepe, Thomas Sikora, Jörn Ostermann, Ebroul Izquierdo, and Touradj Ebrahimi, « Subjective Evaluation of Scalable Video Coding for Content Distribution », *in*: MM '10, Firenze, Italy: Association for Computing Machinery, 2010, pp. 65–72, ISBN: 9781605589336, DOI: 10.1145/1873951.1873981.

[57] Weichao Li, Ricky K.P. Mok, Rocky K.C. Chang, and Waiting W.T. Fok, « Appraising the Delay Accuracy in Browser-Based Network Measurement », *in*: *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, Barcelona, Spain: Association for Computing Machinery, 2013, pp. 361–368, ISBN: 9781450319539, DOI: 10.1145/2504730.2504760, URL: https://doi.org/10.1145/2504730.2504760.

[58] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert Greenberg, and Yi-Min Wang, « WebProphet: Automating Performance Prediction for Web Services », *in*: *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, San Jose, CA: USENIX Association, Apr. 2010,

URL: https://www.usenix.org/conference/nsdi10-0/webprophet-automating-performance-prediction-web-services.

[59] Yi Liu, Yun Ma, Xuanzhe Liu, and Gang Huang, « Can HTTP/2 Really Help Web Performance on Smartphones? », *in*: *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 219–226, DOI: 10.1109/SCC.2016.36.

[60] Jun Lu, *A survey on Bayesian inference for Gaussian mixture model*, 2021.

[61] M-Lab, https://www.measurementlab.net/.

[62] Victor A. Machado, Carlos N. Silva, Rosinei S. Oliveira, Alexandre M. Melo, Marcelino Silva, Carlos R. L. Francês, João C. W. A. Costa, Nandamudi L. Vijaykumar, and Celso M. Hirata, « A new proposal to provide estimation of QoS and QoE over WiMAX networks: An approach based on computational intelligence and discrete-event simulation », *in*: *2011 IEEE Third Latin-American Conference on Communications*, 2011, pp. 1–6, DOI: 10.1109/LatinCOM.2011.6107419.

[63] A Morton, G Ramachandran, and G Maguluri, *Reporting IP network performance metrics: different points of view*, tech. rep., 2012.

[64] *Navigation timing API*, https://www.w3.org/TR/navigation-timing/.

[65] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio 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 '14, Sydney, Australia: Association for Computing Machinery, 2014, pp. 133–140, ISBN: 9781450332798, DOI: 10.1145/2674005.2674991, URL: https://doi.org/10.1145/2674005.2674991.

[66] NDT, https://www.measurementlab.net/tests/ndt/.

[67] 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 18)*, 2018, pp. 217–231.

[68] 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*, IEEE, 2013, pp. 80–89.

[69] First Web page, *World Wide Web*, `https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html`.

[70] PlanetLab, `https://www.planet-lab.eu/`.

[71] SPDY protocol, `https://www.chromium.org/spdy/`.

[72] Feng Qian, Vijay Gopalakrishnan, Emir Halepovic, Subhabrata Sen, and Oliver Spatscheck, « Tm3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking », *in*: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[73] W3C recommendation, *Paint timing API*, `https://www.w3.org/TR/paint-timing/`.

[74] Eric Rescorla, *HTTP Over TLS*, RFC 2818, May 2000, DOI: `10.17487/RFC2818`, URL: `https://www.rfc-editor.org/info/rfc2818`.

[75] Mohammad Rezaei, « Clustering validation », PhD thesis, Itä-Suomen yliopisto, 2016.

[76] A. Ritacco, C. Wills, and M. Claypool, « How's My Network? A Java Approach to Home Network Measurement », *in*: *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, 10.1109/ICCCN.2009.5235346, Aug. 2009, pp. 1–7.

[77] Jan Rüth, Konrad Wolsing, Klaus Wehrle, and Oliver Hohlfeld, « Perceiving QUIC », *in*: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ACM, Dec. 2019, DOI: 10.1145/3359989.3365416.

[78] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin, « A review of clustering techniques and developments », *in*: *Neurocomputing* 267 (2017), pp. 664–681.

[79] Raimund Schatz, Tobias Hoßfeld, Lucjan Janowski, and Sebastian Egger, *DataTraffic Monitoring and Analysis*, Springer-Verlag, 2013, chap. From Packets to People: Quality of Experience As a New Measurement Challenge, pp. 219–263.

[80] scikit-learn, *Clustering: k-means*, https://scikit-learn.org/stable/modules/clustering.html#k-means.

[81] scikit-learn, *Gaussian mixture models*, https://scikit-learn.org/stable/modules/mixture.html.

[82] STEVE SOUDERS, *Velocity and the Bottom Line*, http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html, 2009.

[83] Speedtest, https://www.speedtest.net/.

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

[85] « Subjective quality evaluation VIA paired comparison: Application to scalable video coding », English, *in*: *IEEE Transactions on Multimedia* 13.5 (Oct. 2011), pp. 882–893, ISSN: 1520-9210, DOI: 10.1109/TMM.2011.2157333.

[86] Imane Taibi, Yassine Hadjadj-Aoul, and Chadi Barakat, « Data Driven Network Performance Inference From Within The Browser », *in*: *IEEE ISCC* (2020), pp. 1–6.

[87] Imane Taibi, Yassine Hadjadj-Aoul, and Chadi Barakat, « When Deep Learning meets Web Measurements to infer Network Performance », *in*: *CCNC 2020 - IEEE Consumer Communications & Networking Conference*, Las Vegas, United States: IEEE, Jan. 2020, pp. 1–6, URL: https://hal.inria.fr/hal-02358004.

[88] Stefano Tarantola and Thierry A. Mara, « Variance-based sensitivity indices of computer models with dependent inputs: The Fourier Amplitude Sensitivity Test », *in*: *Int. Journal for Uncertainty Quantification* 7.6 (Apr. 2017), pp. 511–523.

[89] *tcconfig's documentation*, https://tcconfig.readthedocs.io/en/latest/.

[90] Wei-Guang Teng, Cheng-Yue Chang, and Ming-Syan Chen, « Integrating Web caching and Web prefetching in client-side proxies », *in*: *IEEE Transactions on Parallel and Distributed Systems* 16.5 (2005), pp. 444–455, DOI: 10.1109/TPDS.2005.56.

[91] *Tim Berners-Lee*, https://www.w3.org/People/Berners-Lee/.

[92] V. TONG, H. A. TRAN, S. SOUIHI, and A. MELLOUK, « Network troubleshooting: Survey, Taxonomy and Challenges », *in*: *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, Oct. 2018, pp. 165–170, DOI: 10.1109/SaCoNeT.2018.8585610.

[93] Shaun Turney, *Pearson Correlation Coefficient*, https://www.scribbr.com/statistics/pearson-correlation-coefficient/.

[94] IT Union, « Itu-t recommendation p. 800.1: Mean opinion score (mos) terminology », *in*: *International Telecommunication Union, Tech. Rep* (2006).

[95] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki, « EYEORG », *in*: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, ACM, Dec. 2016, DOI: 10.1145/2999572.2999590, URL: https://doi.org/10.1145%2F2999572.2999590.

[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 16)*, Denver, CO: USENIX Association, June 2016, pp. 159–165, ISBN: 978-1-931971-30-0.

[97] Jia Wang, « A Survey of Web Caching Schemes for the Internet », *in*: *SIGCOMM Comput. Commun. Rev.* 29.5 (Oct. 1999), pp. 36–46, ISSN: 0146-4833, DOI: 10.1145/505696.505701.

[98] Tao Wang, Jun Wei, Feng Qin, WenBo Zhang, Hua Zhong, and Tao Huang, « Detecting performance anomaly with correlation analysis for Internetware », *in*: *Science China Information Sciences* 56 (Aug. 2013), pp. 1–15, DOI: 10.1007/s11432-013-4906-6.

[99] Tao Wang, Jun Wei, Wenbo Zhang, Hua Zhong, and Tao Huang, « Workload-aware anomaly detection for Web applications », *in*: *Journal of Systems and Software* 89 (2014), pp. 19–32, ISSN: 0164-1212, DOI: https://doi.org/10.1016/j.jss.2013.03.060, URL: https://www.sciencedirect.com/science/article/pii/S0164121213000721.

[100] Tao Wang, Wenbo Zhang, Jun Wei, and Hua Zhong, « Workload-aware online anomaly detection in enterprise applications with local outlier factor », *in*: *2012 IEEE 36th Annual Computer Software and Applications Conference*, IEEE, 2012, pp. 25–34.

[101] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall, « Demystifying Page Load Performance with WProf », *in*: *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL: USENIX Association, Apr. 2013, pp. 473–485, ISBN: 978-1-931971-00-3, URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/wang_xiao.

[102] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall, « How speedy is {SPDY}? », *in*: *11th usenix symposium on networked systems design and implementation (nsdi 14)*, 2014, pp. 387–399.

[103] 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 16)*, Santa Clara, CA: USENIX Association, Mar. 2016, pp. 109–122, ISBN: 978-1-931971-29-4, URL: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/wang.

[104] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld, « A performance perspective on web optimized protocol stacks », *in*: *Proceedings of the Applied Networking Research Workshop*, ACM, July 2019, DOI: 10.1145/3340301.3341123.

[105] XMLHttpRequest, https://xhr.spec.whatwg.org/.

[106] M. Yajnik, Sue Moon, J. Kurose, and D. Towsley, « Measurement and modelling of the temporal dependence in packet loss », *in*: *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, vol. 1, 1999, 345–352 vol.1, DOI: 10.1109/INFCOM.1999.749301.

[107] Li Yu and Zhiling Lan, « A scalable, non-parametric anomaly detection framework for hadoop », *in*: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, pp. 1–2.

[108] Yasir Zaki, Jay Chen, Thomas Pötsch, Talal Ahmad, and Lakshminarayanan Subramanian, « Dissecting Web Latency in Ghana », *in*: *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 241–248, ISBN: 9781450332132, DOI: 10.1145/2663716.2663748.

[109] Chengwei Zhang, Xiaojun Hei, and Wenqing Cheng, « Performance Evaluation of Web-based Cloud Services in a Browser-Scripting Approach », *in*: KSII Transactions on Internet and Information Systems.10 (June 2016), pp. 2463–2482, DOI: 10.3837/tiis.2016.06.002.

[110] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G Griffin, « Internet routing policies and round-trip-times », *in*: *International Workshop on Passive and Active Network Measurement*, Springer, 2005, pp. 236–250.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Surveillance du réseau basée sur les données Web : de l'estimation de perfromance à la détection d'anomalies

Mot clés : Web,mesures passives, surveillance du réseau, détection d'anomalies,CNN

Résumé : L'objectif de cette thèse est de tirer parti des mesures passives librement disponibles dans le navigateur et des techniques d'apprentissage profond pour inférer la performance du réseau et détecter les anomalies. Nous commençons par déduire les principales propriétés du réseau sous-jacent à partir de mesures de performance Web, en se basant sur des mesures passives obtenues à partir du navigateur. Nous utilisons le Machine Learning pour calibrer les algorithmes qui permettent une telle inférence. En comparant des algorithmes du deep learning à des algorithmes ML classiques comme Random Forest, nous soulignons la faisabilité de la tâche, mais aussi sa complexité, d'où le besoin d'algorithmes d'apprentissage profond sophistiqués tels que les réseaux de neurones convolutionnels (CNN). Ensuite, nous étudions et examinons l'impact de la complexité du Web sur l'estimation de deux paramètres spécifiques, le délai et la bande passante de téléchargement. De plus, nous proposons un framework intégré pour comparer notre approche avec les solutions de surveillance web existantes. Plus tard, nous proposons un système de surveillance réseau original basé sur des modèles de mélanges gaussiens bayésiens (BGMM) couplés à un algorithme pour détecter en temps réel l'apparition d'anomalies.

Title: Web-based data driven network monitoring: from performance estimation to anomaly detection

Keywords: Web, passive measurements, Network monitoring, anomaly detection,CNN

Abstract: The goal of this thesis is to leverage passive measurements freely available in the browser and deep learning techniques to infer network performance and detect anomalies. We start by inferring the main properties of the underlying Network from web performance metrics based on passive measurements obtained from within the browser. We use machine learning to calibrate algorithms that allow such inference. By comparing deep learning algorithms to classical ML algorithms like Random Forest, we highlight the feasibility of the task but also its complexity hence the need for sophisticated deep learning algorithms such as convolutional neural networks (CNN). Then we study and examine the impact of web complexity on estimating two specific metrics, delay and download bandwidth. Moreover, we propose an integrated framework to compare our approach with existing web-based monitoring solutions. Later, we propose an original network monitoring framework based on Bayesian Gaussian Mixture Models (BGMM) coupled with an algorithm to detect in real-time the occurrence of anomalies.