Dissecting Dual-Stack Website Content

Auteur : Dekinder, Florian
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "computer systems security"
Année académique : 2023-2024
URI/URL : http://hdl.handle.net/2268.2/19889

Liège University
Faculty of Applied Sciences

Dissecting Dual-Stack Website Content

Master thesis conducted to obtain the degree of Master of Science in Civil Engineering with a specialization in computer systems security

submitted by

Florian Dekinder

supervised by:

Pr. Benoit Donnet

Mr. Eric Vyncke

Academic Year: 2023-2024

Acknowledgements

I would like to express special thanks to my academic supervisor, Professor Benoit Donnet, for his constant support throughout the year, without whom I would certainly not have dared to submit a paper on our work. I would also like to thank him for providing me with "GenericPlotting", a great plotting facilitator software.

I would also like to thank our corporate contact at Cisco, Mr. Eric Vyncke, for his monitoring of our work and his advice, which enabled us to extend our analyses. I also thank him for providing me with the necessary equipment, without which I would not have been able to carry out the measurement campaign.

My thanks also go to three students from the same cohort: Reda Bellafqih; Haytham Boulaich; Marie Maes, for kindly agreeing to review the paper in depth to identify potential grey areas.

Abstract

LIÈGE UNIVERSITY: Faculty of Applied Science

Dissecting Dual-Stack Website Content

Submitted by Florian DEKINDER

Supervised by Pr. Benoit DONNET and Mr. Eric VYNCKE

Academic Year 2023-2024

The ongoing shift from `IPv4` to `IPv6` is crucial for the evolving Internet landscape. However, the prevalence of dual-stacked environments requires a deeper understanding of how web content is distributed across both protocols, especially given the varied nature of websites configurations. While prior studies have focused on `IPv6` adoption and performance metrics, limited research explores dual-stacked servers' impact on the application layer where user experience is directly affected.

To address this gap, we developed NETQUARTZ, a tool designed to assess the performance and the content delivery of websites across dual-stack servers. We have deployed NETQUARTZ over several vantage points and collected data for more than 200,000 websites, revealing patterns by classifying them into three classes based on server configurations: (*i*) `Fully IPv4` websites, where all resources are loaded from `IPv4`-only servers; (*ii*) `Fully Dual-Stacked` websites, which load all their resources from servers supporting both `IPv4` and `IPv6`; (*iii*) `Mixed` websites, which retrieve resources from a combination of both server types. *Script* and *Image* were identified as the two most dominant categories in terms of content, regardless of the website's class. `Fully IPv4` websites generally contained fewer resources, leading to smaller page sizes and faster load times. `Fully Dual-Stacked` and `Mixed` configuration websites showed better performance under `IPv6`-preferenced loading, despite `Mixed` websites tending to load more resources over `IPv4`.

The thesis is organized as follows: (*i*) we start by introducing the context and motivations behind this work; (*ii*) we position our work with respect to the state of the art; (*iii*) we then provide a detailed explanation of NETQUARTZ and its implementation; (*iv*) we discuss the data collection methodology, explaining the setup and execution of our measurement environment; (*v*) we present our results, first exploring the delivery of web content across websites, we also examine the specific dual-stack content distribution of `Mixed` websites, then assessing broader performance measures within the three website classes and we finally discuss a few topics related to the user experience when browsing a website; (*vi*) finally, this thesis is concluded by summarising its main achievements and suggesting areas for future researches.

Keywords: *dual-stack; `IPv6`; performance; content; NETQUARTZ.*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

Since the early 80's, the Internet Protocol version 4 (IPv4) [19] has been the main protocol supporting packet switching networks and internetworking technology. However, the exponential deployment of the Internet, since the mid-90's, has led to an exhaustion of IPv4 addresses [41], requiring a transition towards its successor, Internet Protocol version 6 (IPv6) [21]. Since then, IPv6 has been more and more adopted [38, 17, 18, 23]. If IPv6 allows for dealing with IPv4 address exhaustion [41], it also comes with an additional feature, called *Extension Header* [22, 13], that leads to more flexibility and innovation. Examples of innovations can be found in observability [10] or forwarding [34].

A smooth transition from IPv4 to IPv6 has been made possible thanks to *dual-stack* devices, i.e., devices with network interfaces that can originate and understand both IPv4 and IPv6 packets [47], avoiding so a complete shift from one protocol version to the other. Up to now, studies primarily focus on the adoption and basic performance of IPv6 metrics, such as addressing and routing [53, 17, 18, 23]. However, few researches focused on dual-stack aspects. For instance, Bajpai et al. [5, 30, 4] provide valuable insights into server performance and transport-layer metrics but do not extend their analysis to the application layer where user experience directly takes place. This is crucial as a significant portion of websites includes content fetched from multiple, distinct servers [11], possibly over different versions of the Internet Protocol, introducing so additional delay. Huston [42, 43] used Google ads to measure the latency of client connections to APNIC servers and found that IPv6 was faster than IPv4 half the time, but experimented with a higher failure rate. Further, Bajpai et al. [38], show a significant reduction in latency for both IPv4 and IPv6 for Alexa top 10K websites. They also report a reduction in IPv6 connection failures and find that the main contributors to these failures are *Image*, *Stylesheet*, *Script* categories. Also, Dhamdhere et al. [23] evaluated these load times on the Alexa list and found that IPv6 performance can be much worse than IPv4 if the AS paths are different, otherwise, they are similar. However, it is unclear which amount of content (e.g., *Image*, *Stylesheet*,

Script, etc.) is delivered through which version of the Internet Protocol.

1.2 Contributions

We make the following contributions: first, we introduce NetQuartz, our tool designed to assess and compare the content delivery capabilities and performance of websites across dual-stack servers. For each website, NetQuartz maintains two loading sessions: (*i*) a full-`IPv4` loading, always preferring `IPv4` connections, and (*ii*) an `IPv6`-preferenced loading, prioritizing `IPv6` connections where available, and defaulting to `IPv4` otherwise. This thesis will carefully explain how NetQuartz has been built to reach those goals.

Second, we have deployed NetQuartz over several vantage points and collected data from more than 200,000 analyzable websites extracted and filtered from `DomCop` list [26].

Third, we propose to classify websites based on their server configurations into three classes: (*i*) `Fully IPv4` websites, where all resources are loaded from `IPv4`-only servers; (*ii*) `Fully Dual-Stacked` websites, which load all their resources from servers supporting both `IPv4` and `IPv6`; (*iii*) `Mixed` websites, which retrieve resources from a combination of both server types. All three classes are initially accessed with full-`IPv4` loading, using only `IPv4` connections. Separately, `Mixed` and `Fully Dual-Stacked` websites are also accessed with an `IPv6`-preferenced loading, which prioritizes `IPv6` connections where possible. For `Fully Dual-Stacked` websites, `IPv6`-preferenced is equivalent to full-`IPv6` loading, as all resources are available on dual-stack servers.

Fourth, whatever the website class, content-related analysis revealed that *Image* and *Script* were the predominant categories, both in terms of the size and the number of resources queried over `HTTP/S`. The analysis also showed that `Fully IPv4` websites typically load fewer resources, resulting in faster loading times. For `Fully Dual-Stacked` and `Mixed` websites, the performance under `IPv6`-preferenced loading consistently outperforms that of `IPv4`. We observed that `Mixed` websites load 20% faster at the median and 11% faster at their 95$^{\text{th}}$ percentile when using `IPv6`-preferenced loading, `Fully Dual-Stacked` websites load 16% faster in median and 24% faster in the 95$^{\text{th}}$ percentile. The observation is particularly interesting with `Mixed` websites, for which we found that 64% of these websites were loading more `HTTP/S` resources over `IPv4` than `IPv6`, yet the overall performance in `IPv6`-preferenced sessions remains more efficient.

A reduced version of this thesis has also been submitted to the ACM Internet Measurement Conference (IMC) 2024, which is documented in the Appendix.

1.3 Organisation

The remainder of this paper is organized as follows: Chatper 2 positions our work with respect to the state of the art; Chatper 3 provides a detailed explanation of NetQuartz and its methodology; Chatper 4 discusses the data collection methodology, explaining the

setup and execution of our measurement environment; Chatper 5 presents our results, first exploring the delivery of web content across websites, then examining the specific dual-stack distribution of `Mixed` websites, and finally assessing broader performance measures; finally, Chatper 6 concludes this paper by summarising its main achievements and suggesting areas for future research. The appendices contain details of technical implementations of NetQuartz, as well as a copy of a paper on our work submitted to the Internet Measurement Conference (IMC) 2024.

1.4 Software artifacts

The various resources of our work are open source.

`Python` NetQuartz code, the generated dataset, and data analysis scripts are available and documented within this gitlab directory: `https://gitlab.uliege.be/Florian.Dekinder/dissecting-dual-stack-web-content`.

Chapter 2

Related works

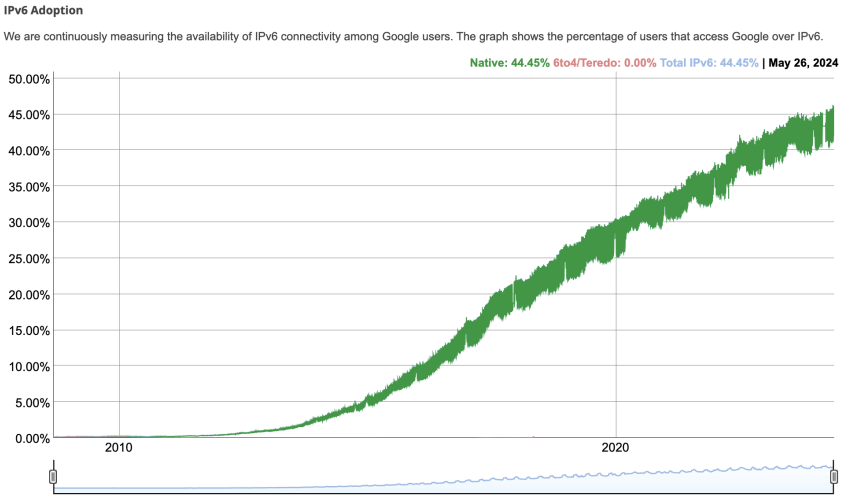
In this chapter, we position our work with respect to the state of the art, analyzing related works and highlighting points of inspiration/divergence relative to our study and the implication on our data collection tool's design: NetQuartz–the latter will be introduced in Chapter 3, we will not go into any more detail here than we did in Section 1.1.

2.1 IPv6 adoption

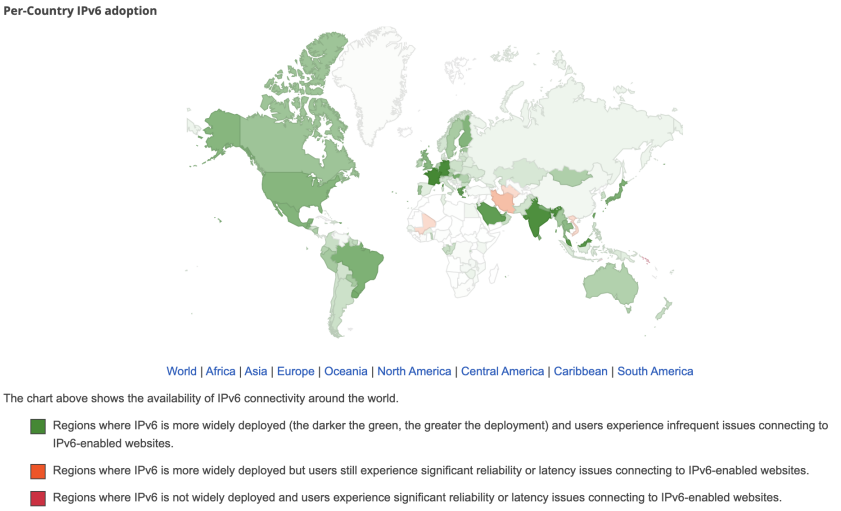
Early studies on IPv6 primarily focused on assessing its adoption, with a focus on addressing and routing aspects [23], with the aim of evaluating the transit technologies used to integrate IPv6 [62, 40] and providing critical insights into the complex dynamics of global IPv6 adoption [53]. More recently, Huston's work (2015) [42, 43], demonstrated through a longitudinal analysis the ineffectiveness of 6-to-4 and Teredo technologies, which have since given way to mainly native IPv6 support. We have reported in Figure 2.1 Google's IPv6 deployment statistics (2024) [38], the figure highlights that 6-to-4 and teredo tunneling technologies have largely disappeared (0.0% remaining), leaving a native IPv6 global adoption at 44.45%. The highest adoption rates are seen in Europe and America, with Oceania, Asia, and Africa following. Recent work [35] (2023) highlights that despite significant progress in IPv6 adoption, with advances in native IPv6 support and a growing number of IPv6-enabled users, heterogeneous adoption rates across industries are still an obstacle to a smoother transition.

2.2 Servers' performance over IPv4 vs. IPv6

Recent research on IPv6 performance typically focuses on transport-layer metrics such as TCP connection time or end-to-end latency. Most studies provide a snapshot of IPv6 performance; longitudinal evaluations of IPv6 performance have been conducted by Huston and Bajpai *et al.*. Huston [42, 43] used Google ads to measure the latency of client



(a) General IPv6 adoption.



(b) Per-country IPv6 adoption.

Figure 2.1: Google's IPv6 adoption statistics [38].

connections to APNIC servers and found that IPv6 was faster than IPv4 half the time, but experimented with a higher failure rate. The most recent longitudinal study to focus on web server measurement is the one performed by Bajpai *et al.* [7] (2019), which focuses mainly on transport layer metrics. Their results show a significant reduction in latency for both IPv4 and IPv6 within Alexa top 10,000 websites. They also report a reduction in IPv6 connection failures and find that the main contributors to these failures (2019) are *Image*, *CSS*, *Script* resources. Older work includes more application-level metrics such as web page load times, Dhamdhere *et al.* [23] (2012) evaluated these load times on the Alexa list and found that IPv6 performance can be much worse than IPv4 if the AS paths are different, otherwise they are similar. Although delay is a sufficient metric to assess performance for smaller web pages, the varying complexity of web services requires an

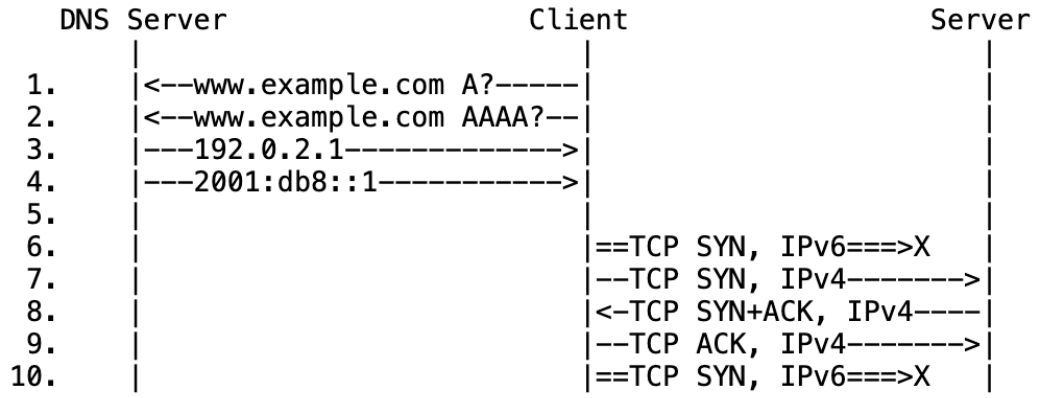


Figure 2.2: Happy eyeballs in action, taken from [67]. The dual-stack client first obtains `A` and `AAAA` `DNS` records, it then starts simultaneously (timer set to 0 ms in this case) one connection over `IPv6` and one over `IPv4`. `IPv6` connectivity appears broken but thanks to the `IPv4` response, the problem will not be felt by the user.

extension analysis at the application level by refreshing previous measurements. Our work aims to provide this extension.

Research on Internet path-length aspects between `IPv4` and `IPv6` has also been performed [37]. It showed that `IPv6` and `IPv4` paths share similar path lengths, implying that, despite `IPv6` 's less stable paths and higher dynamics, the fundamental infrastructure (in terms of distance data travels) is comparable to `IPv4`.

NetQuartz's design was deliberately inspired by metrics explored in all these studies, such as the logging of `IPv4` and `IPv6` failure rates, but also `IPv4` and `IPv6` `RTT` estimation times (but only towards a specific website server, for reasons of time complexity). For the path-length aspect, NetQuartz logged each `IP` address to perform a post-processing geophysical mapping and a rough estimate of the distances covered during full-`IPv4` and `IPv6`-preferenced loading. `traceroute` analysis could arguably be more reliable at this level, but is more complicated to implement due to the number of servers on which the traceroute needs to be performed.

It should also be noted that the longitudinal aspect of certain studies (e.g. Huston's [42]) could, in the future, enable NetQuartz to sketch the evolution of our measurements over months/years. That said, due to the time constraints of this work, we have not yet been able to carry out more than one or two measurement campaigns.

2.3 Happy eyeballs

Happy eyeballs is an algorithm used in dual-stack networks to improve connectivity by reducing delays associated with address family selection (`IPv4` vs. `IPv6`). The algorithm first attempts a connection on `IPv6` and if it is not completed within a certain timer range

(typically 300 ms but this may vary depending on the browser implementing it), it launches a competition between IPv6 and IPv4, with the winner establishing which protocol is selected [67, 58]. An example of happy eyeballs in action is shown in Figure 2.2.

Recent research (2016–2019) has evaluated the impact of happy eyeballs on performance, showing that the default 300 ms timer was not always leading to the best outcome [6, 7]. Piraux *et al.* [55] recently provided a DNS extension as an adaptative address family selection algorithm able to dynamically select, according to end-to-end latency measures, the best address family for establishing the connection.

Happy eyeballs has greatly contributed to the reduction of protocol selection times, but it will not have a significant impact on the design of NetQuartz for two reasons: (i) the happy eyeballs algorithm is complicated to implement in practice (in its optimality), (ii) we seek to compare IPv4 and IPv6 in a straightforward way and not the efficient selection of one of the protocols.

2.4 Web content delivery

Web content has also been examined. Butkiewicz et al. [11] found that a significant portion of web pages includes content fetched from multiple, distinct servers, introducing additional delay. They highlight that it is not just the total size of the page that affects load times but, more significantly, the number of individual objects requested. Advertising content was also captured: Yutian *et al.* recently (2023) developed AdHere for this purpose, an automated technique to detect and repair intrusive ads [68]. Other works [45, 11] show that images and JavaScript are among the prominent content types, with JavaScript objects, in particular, contributing substantially to the total page size. Most of the studies were only focused on the root home page of the service.

On the contrary, Lookyloo [6] – a networking tool web interface – focuses on the classification of all the resources linked to the home page along with their category but, to our best knowledge, no paper has been published on it. Bajpai et al. [30, 3] give insights at the content level of dual-stack web servers, their focus was mainly on the failure rate and the performance of content delivery over IPv4 and IPv6. Our work seeks to complement theirs by delving into application-level metrics and evaluating how much of the content is distributed over IPv4 and IPv6.

When discussing web performance, user experience is a crucial factor. Sengupta et al. [60] conducted a study using Google's Chrome User Experience Report (CrUX), which focused specifically on user experience and user-related metrics, particularly those associated with web page rendering. Although this research is orthogonal to our work, it underscores disparities in these metrics across devices and geographic regions, revealing that desktops generally outperform mobile devices. Furthermore, it highlights that certain countries benefit from superior network performance, leading to a better overall user experience.

2.5 Website content extraction

This section is mainly dedicated to a single study: *Web Data Extraction, Applications and Techniques: A Survey* by Ferrara *et al.* [31] (2014).

This study presents the different approaches to consider when capturing web content and, in particular, allowed us to select the best capture method for our needs. These techniques include: (*i*) the use of regular expressions on simple `HTML` documents; (*ii*) natural language processing techniques (`NLP`) combined with information extraction (`IE`) principles; (*iii*) browser simulation techniques combined with `HTTP/S` request listening. There are many other techniques of varying complexity presented in the study, but this last one is particularly interesting and is the one we will be applying to our work using `Selenium` technology (introduced later in Chapter 3). In fact, the most effective way to simulate a web environment is to simulate a browser by itself, because that is where all the complexity lies. The browser mainly takes care of parsing the `HTML` document and identifying the resources to be queried, then we only need to listen and log all the `HTTP/S` requests generated by the browser. Within Google `Chrome` browser, this listening is made possible by the `Chrome` Performance Tool (or *DevTools*) [16], which we will briefly introduce in Section 3.1.

2.6 What we bring to the table

In conclusion, all the related work shows a gap in the literature when it comes to dual-stack content distribution. In other words, many studies remain fixed at the level of a single server, at the transport layer. To the best of our knowledge, no studies delve into the complexity of the `IP` configuration of the servers that make up a website. The latter is a major failing because it somewhat distorts the real purpose of a website, which is to download content from a multitude of different origins. An important piece of information is missing: which fraction of content is delivered under which protocol, and which content category (*Document*, *Image*, *Script*, etc.). This is the first gap our study aims to fill.

Secondly, `IPv4` vs. `IPv6` performance studies (still limited to the scale of a single server) are starting to look dated. Our work will not only bring these statistics back into the picture, but will also consider them in the more complex context of websites, hiding a multitude of web servers, and attempt to explain these statistics.

We will see later in Chapter 5 that this opens the door to a new way of looking at websites, by classifying them according to the nature of the servers that make them up: `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed` websites. In particular, we have identified patterns within these classes: websites made up of `IPv4`-only servers seem faster at first sight, but they are also smaller in size. `IPv4` is not the reason for their speed; this new consideration allows us to capture the web in its entirety while avoiding hasty conclusions about the network layer.

Chapter 3

NetQuartz: the Data Collection Tool

3.1 NetQuartz: a network analyzer providing transparent results

This chapter presents NetQuartz, a `Python` network analyzer software, designed to give a transparent view of the web's underlying infrastructure connections, by providing an analysis of both `IPv4` and `IPv6` protocol used while loading websites. NetQuartz is adopted in this study because of the need to evaluate and compare data contents and delivery performance over the dual-stack servers. The later section will describe the comprehensive approach that the tool used which involves data collection, processing, and analysis which together will highlight the understanding of content delivery across the different web categories (*Image*, *Script*, etc.), as well as the experience of the user. We will first look at some of the technical challenges posed by the underlying nature of NetQuartz (Section 3.2), then present its general working overview (Section 3.3). Section 3.4 will also describe the NetQuartz hyperaprameters, the output format ((Section 3.5)) will also be highlighted. We will finally showcase some practical details of NetQuartz implementation (Section 3.6).

3.2 Technical challenges

3.2.1 Capturing resources

Capturing the entire spectrum of web resources is a significant technical challenge due to how modern websites operate. The traditional method of `HTML` parsing [64], the standard technique for the identification of web resources, is limited for modern websites. That is because the `HTML` parser retrieves only the standard elements of a webpage and will not work for any dynamically loaded content. Modern websites heavily employ dynamically loaded content that is fetched using JavaScript in the background. This content is not loaded when parsing the webpage with the standard `HTML` parser, making it challenging

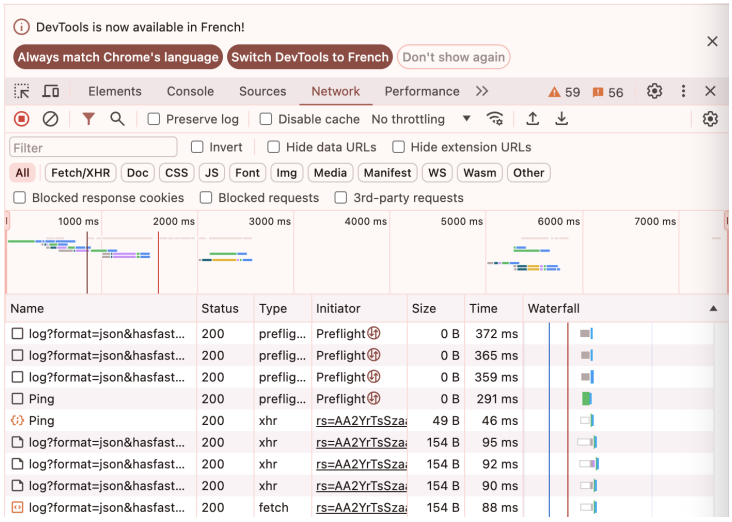


Figure 3.1: `Chrome` network performance tool output on google.com

to collect the information. To solve this problem, NETQUARTZ includes an automated browser, which uses `Selenium` [59] with a `Chrome` driver [25]. NETQUARTZ interacts with the webpage as any human user (through a browser) would, allowing the survey of all elements of the webpage, including asynchronous requests. `Selenium` is a powerful automation browser framework that imitates a user, allowing a bot to gather information concerning every resource rendered throughout a browsing session. We will use the `Chrome` network performance tool [16], through `Selenium`, to capture resource categories such as *Image*, *Script*, *Media*, *Document*, *Font*, *Stylesheet*, and resources dynamically loaded through the *Fetch & XHR* API. Figure 3.1 highlights the output of the `Chrome` performance tool, through the *network* window, while loading the Google web page. We can see that we have access not only to resource categories but also to timing and `HTTP` metrics, typically

- The `URL`
- `HTTP` response/request header
- `HTTP` status code
- `DNS` time spent for the resource
- `SSL` time spent for the resource in case the resource has been loaded over `HTTPS`
- The Time-To-First byte sent by the server (`TTFB`)
- The total time needed to download the resource
- The category of the resource (*Image*, *Script*, etc.)

However, as we will see in the next section, we will not be using these metrics to compare `IPv4` and `IPv6` website loading. We will only keep the type/category (*Image*, *Script*, etc.) along with the `URL`.

3.2.2 Comparing IPv4 and IPv6 protcols

This challenge can be divided into several sub-challenges.

- First of all, why not keep selenium and the Chrome performance tool to compare IPv4 and IPv6 performance metrics? Because it is complicated in practice, programmatically, to force a browser to use IPv4 or IPv6. This would also include a double Selenium session per website, with a lot of unnecessary processing (website rendering, etc.). We therefore opted for Python's *requests* library [9], well known for its implementation of the HTTP/1.1 protocol, and support for TCP session persistence.
- An effective analysis of how modern websites operate requires the development of a download protocol that adapts to the complexity and size of a website. While prior studies have operated on a (dual-stack) server scale, doing this when the website fetches its resources from a wide range of servers creates another level of complexity. For this reason, we introduce IPv6-preferenced loading. IPv6-preferenced loading is such that all the resources on dual-stack servers should be accessed using IPv6 and the resources hosted on IPv4-only server should be accessed using an IPv4 connection. This approach enables NetQuartz to compare the two major IP families when operating on a website scale.
- We must then opt for an efficient selection of performance metrics, considering that the initial objective is to compare IPv4 and IPv6, and not only performance but also content delivery. We will need to include the size (in bytes) of each resource, the loading time (from request initiation to complete download), RTT estimates (on well-defined servers such as the main HTML hosting server), and the ability to log resource failure rates over both IPv4 and IPv6-preferenced sessions. We should also note that to capture the protocol's contribution to the website loading, we **have to** log the IP addresses used to establish the connection towards each resource.

3.3 NetQuartz, a general overview

We will now describe the practical development of NetQuartz, carefully designed taking into account the various challenges introduced in Section 3.2.

NetQuartz, our tool for measuring content delivery and performance of websites, works in three steps as illustrated in Figure 3.2:

- Main service domain processing (Section 3.3.1). It collects information such as the certificate being served by the domain and the number of A and AAAA DNS records associated with the main service;
- Main HTML domain processing (Section 3.3.2). It performs a DNS lookup for the domain and logs the number of A and AAAA DNS records associated with this server, it also sends ping towards the server hosting the main HTML of the website;
- Resources collection (Section 3.3.3). It collects (through a headless browser tech-

nology) and logs all the resources and their categories (*Image*, *Script*, etc) that are loaded along with the main HTML document;
- Website loading comparison protocol (Section 3.3.3). It loads the full website both in a full-IPv4 and an IPv6-preferenced loading and collects metrics from the loading of each resource.

3.3.1 Step 1: Main service domain processing

As depicted in Figure 3.2a, NetQuartz begins by accepting a list of domains along with the URLs of a web home page that can be queried on that domain and the HTTP status code returned while fetching the home page. The domain of these URLs points to the server that hosts the main HTML document of the website. Only entries with a successful status code (200) are considered for processing.

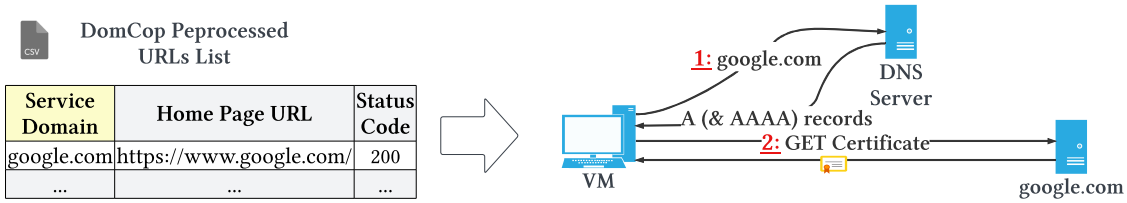
For each service domain in the list, NetQuartz starts by querying the DNS for retrieving associated A and AAAA records. In addition, NetQuartz fetches the certificate associated with the domain. The certificate will be used to determine how many of these websites are still delivering an expired certificate and try to correlate this breach to potential defects in terms of loading times.

3.3.2 Step 2: Main HTML domain processing

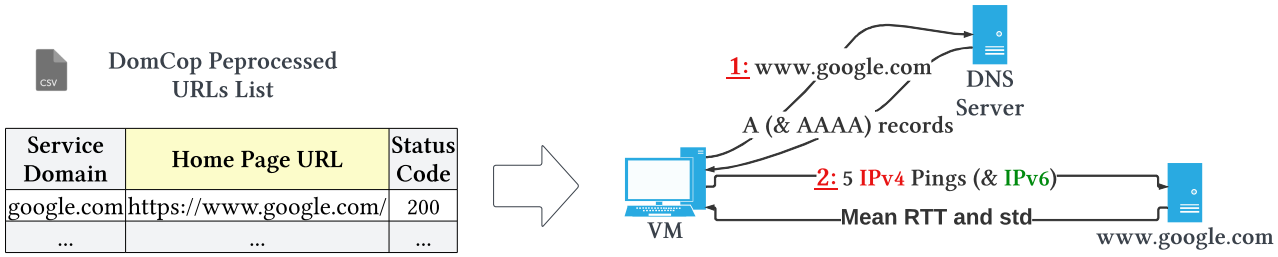
After extracting information from the main domain, we will now focus on the domain referencing the server hosting the main HTML of the website because the two are often different and the main domain may not be sufficient to capture the entire spectrum of the website. Figure 3.2b shows that metrics extraction from this step includes similar DNS record counts (A and AAAA), and then NetQuartz perform ping in IPv4 (and IPv6 if the server has been identified as dual-stack) on the first record returned by the DNS. NetQuartz logs the average time and standard deviation associated with these pings, sending a predefined number of ICMP packets to each home page's hosting server. The goal is to have an estimation of the RTT (in both IPv4 and IPv6 when available) towards this main server, it will thus be possible to determine at a later stage whether this metric is a sufficient indicator of website performance.

3.3.3 Step 3: Resources collection

Most websites not only load the main HTML for the home page but will also parse it [64] to load additional resources (*Image*, *Script*, etc) so that they can ensure the website's complete rendering. NetQuartz relies on Selenium [59], a headless browser technology, for facilitating the collection of these resources. Figure 3.2b illustrates this step: we use a Chrome browser through a dedicated executable driver [25], allowing script-driven browser interactions. The communication between Selenium and the chromedriver is handled

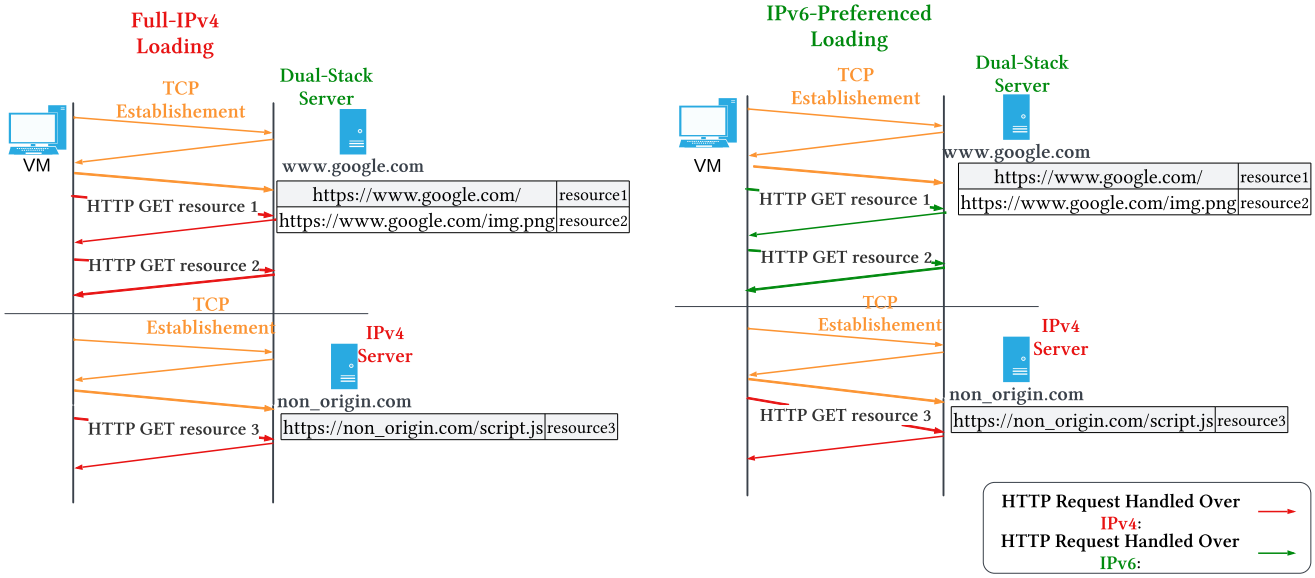


(a) Step 1: Main service domain processing.



(b) Step 2: Main `HTML` domain processing.

(c) Step 3: Resources collection.



(d) Step 4: Website loading comparison protocol.

Figure 3.2: NetQuartz general overview.

through the WebDriver protocol, an HTTP-based protocol that uses JSON objects to instruct the browser what actions to perform [61]. The different resource categories that have been captured are: *Image*, *Script*, *Media*, *Document*, *Font*, *Stylesheet*, and resources loaded through the *Fetch & XHR* API. The cookie mechanism used by websites is a constraint on our resource collection because as long as cookies have not been accepted, not all resources will be captured. To remedy this, we are taking advantage of the simulated Chrome browser to add a dedicated Chrome extension: *I still don't care about cookies* [63]. All captured resources are kept in memory for further analysis.

3.3.4 Step 4: Website Loading Comparison Protocol

Before establishing the methodology for comparing websites, it is worth looking at the generic behavior of a dual-stack user when loading a website from a set of servers. The browser first performs a DNS lookup on the domain hosting the main HTML document, the latter is then retrieved via an HTTP/S request. As soon as the main HTML is available, the browser parses it to identify all the resources needed to render the web page, which may come from servers different than the one hosting the main HTML. When resources are available in both IPv4 and IPv6, the default behavior of most browsers is often to prefer IPv6 [62]. This can be problematic if the IPv6 connectivity is broken. It is for this reason that most browsers now implement the Happy Eyeballs algorithm, which recommends selecting one address family to establish the connection, as a result of a competition started after a specific timer (typical value is 300 ms for Chrome and Firefox) [67, 58]. For practical reasons and to focus only on the comparison between IPv4 and IPv6 and not on an efficient selection of the protocol, the Happy Eyeballs algorithm is not included in our protocol.

Figure 3.2c catalogs our protocol for comparing website loading times, highlighting the complexity of web resource downloads, which often originate from servers distinct from the origin domain. The protocol takes as input the list of resources collected with Selenium during step 2 (Section 3.3.2) and mimics as much as possible a user browsing behavior, adhering to the HTTP standard [32, 33] and connection persistence (for practical reasons, a single TCP connection is persisted). Sequential requests through session management showcase sessions with full-IPv4 loading or an IPv6-preferenced loading, illustrating that prioritizing IPv6 does not guarantee exclusive IPv6 resource loading, as depicted in the figure. To mitigate biases between the two loading strategies, a DNS query precedes each HTTP/S GET, ensuring domain name server caching before the loading times measurements.

During each HTTP/S GET request, additional metrics are collected, such as the resource URL and the *content-type* HTTP header. For both full-IPv4 and IPv6-preferenced loadings, we measure the load time of each resource, we also collect the byte size of the HTTP/S response, the status code of the response, and information about any resource download errors. All the data is logged into a json file and kept for further postprocessing. The total website loading time is computed by summing the loading times of all successfully

loaded resources within both `IPv4` and `IPv6`-preferenced loadings, ensuring neither loading strategy is unfairly penalized for experiencing more failures or timeouts. To preserve data integrity at this level, failure reasons are logged for later comparison, aiming to analyze the failure rate (and reasons) of web resources within full-`IPv4` loading versus `IPv6`-preferenced loading.

NetQuartz proactively identifies and handles specialized `URLs` [65], such as `data URL` that encodes data directly within the `URL` itself [51], these resources do not involve traditional network loading over `IPv4` or `IPv6`. They are still logged for postprocessing (Section 3.3.4), including the assessment of the size and category of data encapsulated within `data URL` resources.

3.4 NetQuartz hyperparameters

The first hyperparameter–related to main `HTML` domain processing (Section 3.3.2)–involves the count of `IPv4` (and `IPv6` if applicable) `pings` performed without compromising the tool's performance. The value chosen for this parameter will be discussed later in Section 4.3.

The majority of websites dynamically load a significant portion of their content via the *Fetch* API and/or the XMLHttpRequest (*XHR*) API [45]. This introduces a degree of ambiguity to the concept of a website's loading process, theoretically allowing it to extend indefinitely. To detect full initial loading, `Selenium` uses the browser's "*load*" event–a pivotal moment in a webpage's lifecycle signaling the full loading and parsing of the page [48]. This will not take into account content dynamically loaded post-initial "*load*" event via asynchronous calls.

That brings us to our second hyperparameter: we have implemented a timer "`selenium_timeout` " for the web loading process conducted by `Selenium` (Section 3.3.3). Similarly, another timer "`resource_timeout` " is used within the fourth NetQuartz step (Section 3.3.4), applying to the loading time of each resource so that we prevent resources from loading indefinitely or in case the targeted web server is not even responding. Upon expiration, this timer exposes the timeout's cause and context: a "*connect timeout*" denotes a failed server connection attempt, whereas a "*read timeout*" occurs upon a failed read attempt from an already established connection.

Another critical parameter is the data write rate to disk, introducing a protective layer for the tool by logging data at a specific frequency. The last parameter relates to a more global timeout "`processing_timeout` " that limits the total NetQuartz processing time for each domain. This is intended to prevent excessive time spent on sites with an abundance of resources that exceed a predefined threshold. If the total time exceeds the set limit, domain processing stops, but data collected is kept, ensuring usability while acknowledging the incomplete analysis of all site resources. The values chosen for these parameters will be discussed later in Section 4.3.

3.5 NetQuartz output format

Each instance of NetQuartz will eventually generate a `json` file summarizing all the information it has extracted for all the domains given as input. Figure 3.3 shows an example of an output format for the facebook.com domain, including in particular

- *main_page*: the `URL` pointing towards the main `HTML` *Document* resource.
- *selenium_load_time*: the `Selenium` load time, which corresponds to the time spent in the third's NetQuartz step (see Section 3.3.3)
- *status* indicating how the NetQuartz processing took place: (*i*) 200 means all went well; (*ii*) -1 means a `selenium_timeout` has occurred; (*iii*) -2 codes mean there was a network error during processing; (*iv*) -3 means the `processing_timeout` has occurred, such that the processing of this domain has been manually stopped, but with a fair amount of resource between full-`IPv4` and `IPv6`-preferenced sessions; (*v*) -4 codes are intended for very unusual cases where a popup on the website cannot be closed, leading to a `Selenium` crash.
- *a_service*, *aaaa_service*, *a_mainPage*, *aaaa_mainPage* are respectively fields indicating the number of `A` and `AAAA` records associated to: (*i*) the main service domain (facebook.com in our case); (*ii*) the domain pointing towards the main `HTML`'s hosting server (www.facebook.com in our case).
- *ping_avg_v4*, *ping_avg_v6*, *std_ping_v4*, *std_ping_v6* are the average `RTT` estimations and the standard deviations, over both `IPv4` (and `IPv6` if applicable) `pings` performed on: (*i*) the main service domain (facebook.com in our case); (*ii*) the domain pointing towards the main `HTML`'s hosting server (www.facebook.com in our case).
- *cert_info* is a dictionary including certificate validity interval limits of the main service domain (facebook.com in our case) and a `Python` timestamp representing the precise moment when the certificate was fetched.
- *time_taken* is simply the total NetQuartz processing time spent on that whole domain.
- *resources* is an array of all the resources of the website, it includes metrics over both full-`IPv4` and `IPv6`-preferenced session, we use *"v4_Sess"* and *"v6_Sess"* suffixes to identify them respectively. We therefore have the following session-dependent metrics for each resource:: the `HTTP` status code, the `IP` address used during connection establishment, the load time, the size (in Bytes), *info_vxSess* contains resource failure reasons: connection or reading timeouts. We should note that *ip_v6Sess* can be an `IPv4` address if the targeted server is `IPv4`-only (see `IPv6`-preferenced loading in Section 3.2.2). Some session-independent fields are also included such as: (*i*) the `URL` of the resource (collected with `Selenium`) (*ii*) *uri_info* which contains the `MIME` type if the resource is following the `data URL` encoding scheme; (*iii*) the `HTTP` *content-type* header; (*iv*) the *type* of resources inferred by `Selenium` (through `Chrome` performance tool).

```
{
"facebook.com": {
    "main_page": "https://www.facebook.com/",
    "selenium_load_time": 1.2073695659637451,
    "status": 200,
    "a_service": 1,
    "aaaa_service": 1,
    "a_mainPage": 1,
    "aaaa_mainPage": 1,
    "ping_avg_v4": 27.307,
    "ping_avg_v6": 31.864,
    "std_ping_v4": 0.005,
    "std_ping_v6": 0.029,
    "cert_info": {
        "notBefore": "20231125000000Z",
        "notAfter": "20240223235959Z",
        "fetch_time": 1708021401.6429517
    },
    "time_taken": 9.799227952957153,
    "resources": [
        {
            "url": "https://www.facebook.com/",
            "status_v4Sess": 200,
            "status_v6Sess": 200,
            "ip_v4Sess": "31.13.81.36",
            "ip_v6Sess": "2a03:2880:f116:83:face:b00c:0:25de",
            "load_time_v4Sess": 0.35506606101989746,
            "load_time_v6Sess": 0.33942675590515137,
            "size_v4Sess": 100000,
            "size_v6Sess": 100000,
            "uri_info": null,
            "info_v4Sess": null,
            "info_v6Sess": null,
            "type": "Document",
            "content_type": "text/html; charset=\"utf-8\""
        }, //... more resources
    ]
}, //... more domains
}

```


Figure 3.3: NetQuartz `json` output format

3.6 Implementation notes

3.6.1 Patching for address family selection and connection details

Two significant patches were applied to the `Python` standard *socket* [56] library and the *urllib3* [50] library. These modifications were crucial for forcing the use of `IPv4` and for capturing the `IP` addresses used during the connection establishment because they are not supported by default by these libraries, even though they are low-level. See Appendix A for detailed implementation.

3.6.2 Timing network requests

The standard `HTTP` GET timeout in the *requests* library is designed to trigger based on **each** response from the server. While this is useful for assessing server responsiveness, it does not account for the total time required to download the entire response. The problem is that the timer resets itself and then applies to each TCP exchange involved within the fetching. Let us look at an example to clarify this: if this timer is set to 2 seconds, and the server begins responding before this limit and then transmits **one** byte of data every 1.999 seconds, the timer will never be triggered and the download can take a relatively long time (even infinite, we have observed such resources that never finish downloading and "refresh" at a certain frequency). This is a problem because NetQuartz is designed to always complete its processing in a non-infinite time.

To address this, we implemented a custom timing mechanism using `Python`'s *sys.settrace* [57] function to monitor and enforce a timeout based on total download time, not just the initial response. This method involves setting a trace that periodically checks the elapsed time during the download process against our `resource_timeout`. For a detailed implementation please refer to Appendix A.

The trace function in `Python`, used to implement the custom timer, executes a check only at each line of execution within the `HTTP` GET function. Given that the network request function spends most of its execution time waiting for I/O operations, the relative `CPU` time used by the trace function is negligible (and the same goes for the overhead introduced). See Appendix A for detailed implementation.

Chapter 4

Data Collection Methodology

Figure 4.1 provides an overview of the methodology followed when running NetQuartz for collecting data. As illustrated, a four-steps methodology is applied, each of those steps being described in the following sections.

4.1 Step 1: Domains List Collection

Due to the discontinuation of the `Alexa` list, we rely on `DomCop` [26] which lists domain services (e.g., google.com). The list contains 10 million entries but we limit ourselves to the first 300,000 websites for the processing and analysis (list obtained on February 19th, 2024). As explained in Section 3.3.1, NetQuartz needs as input not only the main service domain but also the web home page `URL` for that service, along with the `HTTP` status code fetched. These pieces of information are not included by default in the `DomCop` file, thus requiring a preprocessing stage (Section 4.2).

4.2 Step 2: Preprocessing Step

An initial `HTTPS` connection attempt to each website is made, following `HTTP` redirection codes (range 300 to 399) until a successful 200 status code is obtained. This redirection strategy is commonly employed by web services to guide users directly to the service's web home page (e.g., navigating to google.com in a browser redirects to https://www.google.com/). If the `HTTPS` attempt fails, a subsequent attempt over `HTTP` is made. This process allows us to determine the applicable application layer protocol, the `URL` for accessing the service's home page (e.g., `https://www.google.com/`), and its status; all such information is appended to the original `csv` file.

The preprocessing phase results are summarized in Table 4.1: 73% of input websites yielded a home page with a 200 (OK, request succeeded) status code, 21% responded with error status codes, while 6% did not respond to connection requests. Despite a preference for `HTTPS` connections, 5% of the websites are still exclusively accepting `HTTP` connections.

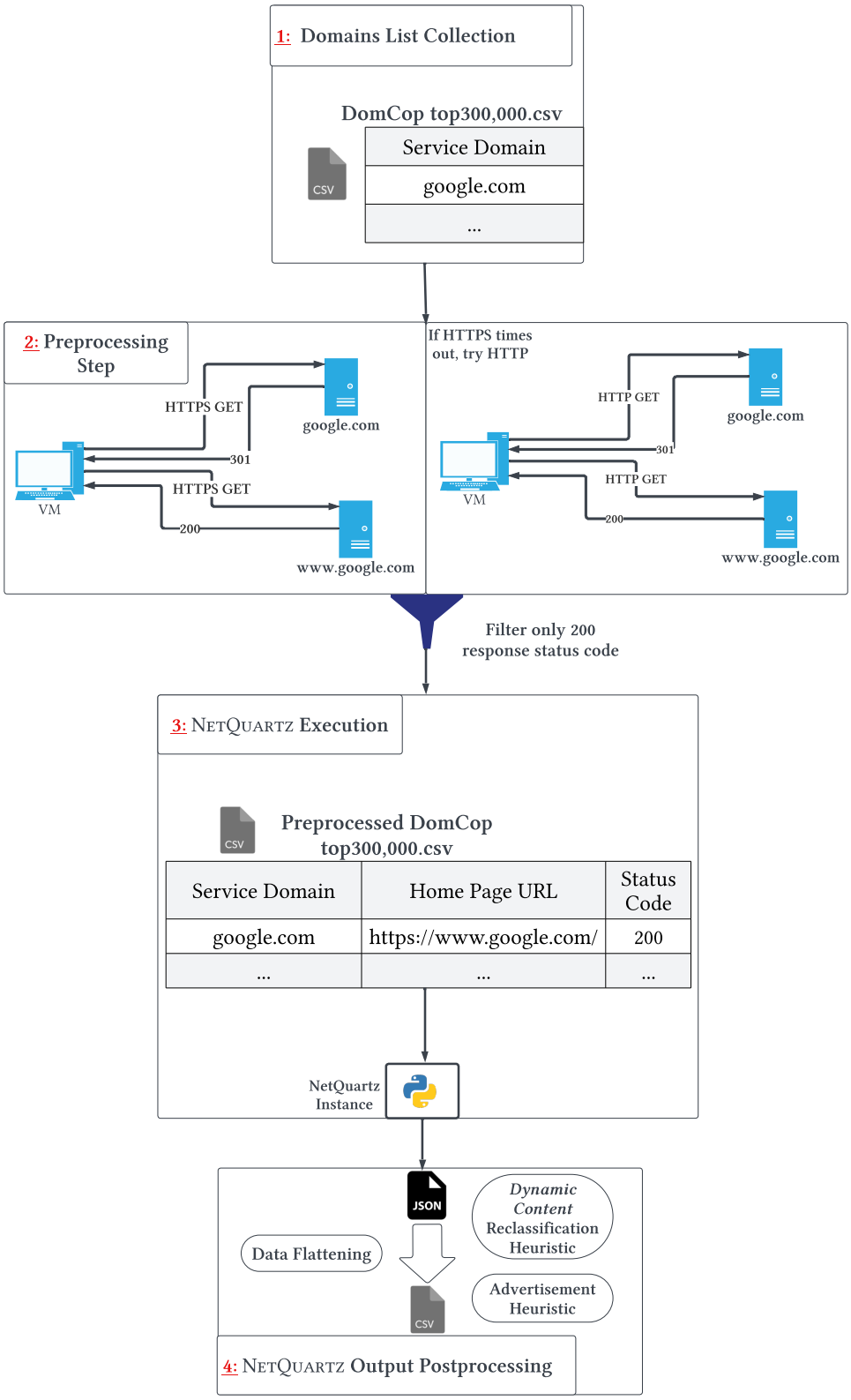


Figure 4.1: Data collection methodology overview.

Category	**Total**		`HTTP`		`HTTPS`	
	Raw	Prop.	Raw	Prop.	Raw	Prop.
All Websites	300,000	1.0	16,398	0.054	283,602	0.946
200 (Success) Status Code	217,973	0.727	9,264	0.031	208,709	0.696
Status Code $\neq$ 200	63,322	0.211	6,012	0.02	57,310	0.191
No Response	18,705	0.062	/	/	/	/

Table 4.1: Summary of the preprocessing step (total: 300,000 websites).

4.3 Step 3: NetQuartz Execution

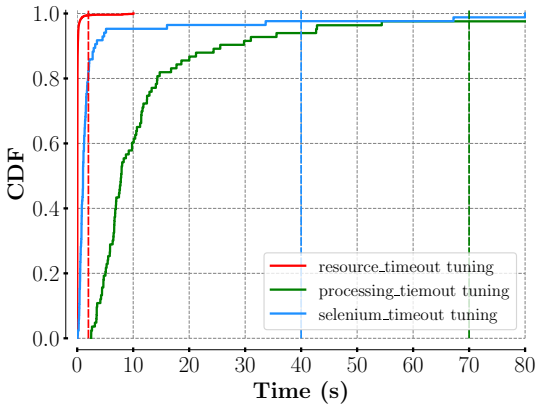
We deployed NetQuartz on three vantage points (VP) that are hosted by different cloud service providers across three regions, one in North America hosted by *Digital Ocean* [24], one in Europe hosted by *OVH* [54], and one in Asia hosted by *Vultr* [66]. Each VP was tasked with allocated portions of the `csv` resulting from previous steps: North America and Asia (both got 1 `CPU` and 0.8 `GB` of `RAM`) dealt each one with 25,000 randomly selected items from the `csv`, while the European VP (8 `CPUs` and 32 `GB` of `RAM`) run through 250,000 websites randomly picked up from the `csv`. The measurement campaign was launched on March 2nd, 2024, and concluded on March 25th, 2024.

The values of each NetQuartz's hyperparameter (Section 3.4) were fixed based on the observation of Cumulative Distribution Functions detailed in Figure 4.2. The tuning is based on 100 websites randomly sampled from the top 300,000 websites in `DomCop`'s list. `selenium_timeout` (with x-axis being the time spent loading the website with selenium) is fixed to 40 seconds, to capture 95% of the websites from the tested sample. Similarly, `processing_timeout` (with x-axis being the total time spent by NetQuartz to process the website) is fixed to 70 seconds. `resource_timeout` (with x-axis being the time spent loading the web resource) is intentionally set to 2 seconds to capture 100% of resources, the goal is to prevent resources from loading indefinitely, we still want to capture a maximum amount of them. The number of `IPv4`– and `IPv6` when applicable – `pings` is set to 5. Figure 4.2b shows that with 5 pings, we still get a reliable snapshot of the network's conditions without significantly increasing the overall processing time per domain, ensuring that NetQuartz remains efficient in processing the top300,000 websites from `DomCop`.

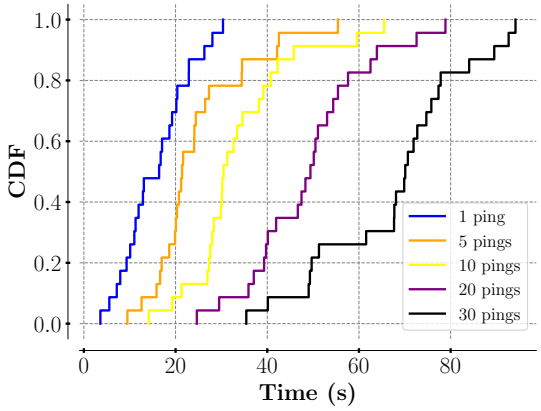
Fixing the number of NetQuartz parallel instances to run is highly dependent on the measurement probe, we recommend performing an analysis of website **load times** to ensure that no bias is introduced. Figure 4.2c shows the results of tests carried out from *OVH* European machine and focusing on the first 20 websites of `DomCop`'s list. Loading time stability seems assured up to 8 instances; to lighten the VM, we deliberately allocate half of them to the *OVH* vantage point. North America and Asia, less powerful, both ran 1 instance of NetQuartz.

Processing Completed		`selenium_timeout`		`processing_timeout`		**Internet Errors**		**Unclosable Popups**	
Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.
202,845	0.932	3,302	0.015	11,395	0.053	0	0.0	431	0.0

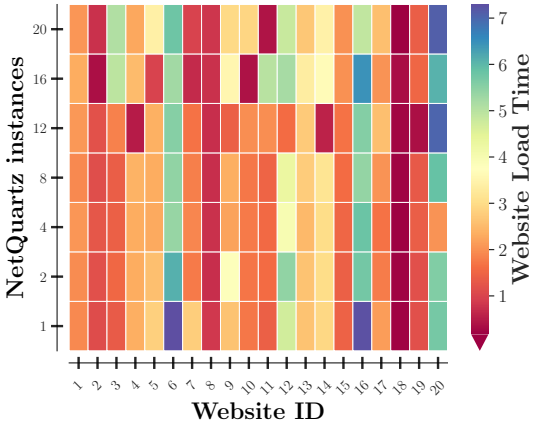
Table 4.2: Summary of the NetQuartz execution step (total: 217,973 websites).



(a) Tuning of the timer values. Dotted lines highlight the chosen timer value.



(b) Tuning of the number of pings to perform, x-axis is the total processing time spent on a website by NetQuartz.



(c) Tuning of the number of NetQuartz parallel instances.

Figure 4.2: NetQuartz hyperparameter tuning.

NetQuartz execution summary is reported in Table 4.2 which indicates that of the 217,973 websites accepted by the preprocessing stage, 93.2% encountered no errors, 1.5% triggered the `selenium_timeout`, and 5.3% had their processing manually halted upon exceeding the `processing_timeout` though these sites remain analyzable (see Section 3.4), with an equitable assessment of their resources between full-`IPv4` and `IPv6`-preferenced sessions.

4.4 Step 4: NetQuartz Output Postprocessing

When NetQuartz processing is done on each VP, `json` outputs are merged, parsed, and flattened into a final `csv` file. During this postprocessing stage, we apply a content reclassification heuristic to fill a gap in the classification of the `Chrome` performance tool (Section 4.4.1). In addition, we take advantage of this postprocessing step to push content classification, trying to derive a lower bound on the ads employed by the website and their content categories (methodology described later in Section 4.4.2). NetQuartz has not been programmatically designed for total ad capture–as this would require, among other things, simulating a user scroll down on `Selenium`. This stage is also responsible for processing `data URL` resources logged by NetQuartz (Section 3.3.3) to infer their types and sizes.

4.4.1 Reclassifying Resources Loaded Through *Fetch & XHR* API

While straightforward `HTML` document parsing techniques might suffice when classifying static content, they fall short in capturing the full spectrum of dynamic web content [31]. `Selenium` makes us able to operate the `Chrome`'s network performance tool, accessible through the `Chrome` driver interface, to infer the following categories: *Document*, *Image*, *Media*, *Fetch*, *XHR*, *Font*, *Stylesheet*, *Script*, and *Other*. That last category is a kind of tie-break, for resources that do not fall within one well-known category.

`Chrome`'s network performance tool does not explicitly reveal resource categories for *Fetch & XHR*, they represent more a classification of requests within the dynamic context of a site. This categorization gap comes from the `Chrome` performance tool's dual consideration of content type and loading context, with dynamic loading contexts (*Fetch & XHR*) taking precedence over content-based categories.

Addressing this, we use the *content-type* header collected by NetQuartz (Section 3.3.3) to build a heuristic able to reclassify the *Fetch & XHR* context-related categories into the content-related ones. The *content-type* `HTTP` Header reliability is not without its challenges; services might declare a content type that diverges from the actual content, a practice countered by browsers through `MIME` (Multipurpose Internet Mail Extensions) type sniffing [27]. This involves payload inspection – following well-defined standards [1] – to derive the true content type, potentially influenced by the resource's file extension. Ethically, our developed heuristics refrain from inspecting payload content.

The basis of the heuristic is keyword detection within the *content-type* header encoding the resource `MIME` type, described by type, subtype, and parameters:

```
type/subtype;parameter=value
```

Keyword detection focuses on the type and subtype fields, the process is summarized in Figure 4.3.

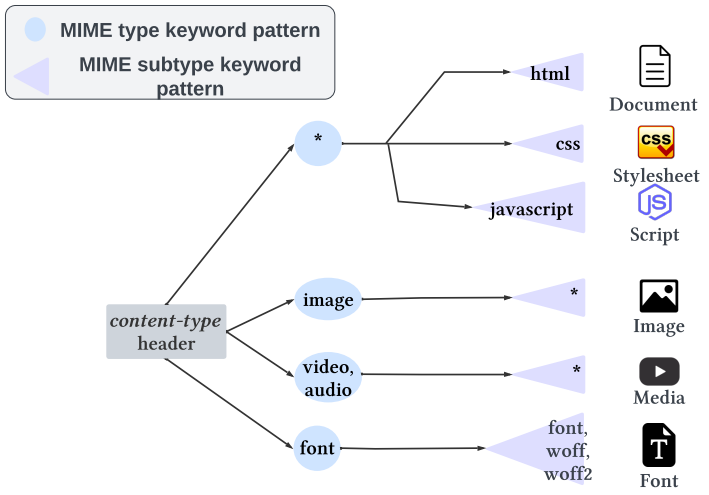


Figure 4.3: Resource classification heuristic, based on the *content-type* header of the `HTTP` response. Asterisks ("*") indicate no particular emphasis on the field.

Type	True Positive		False Positive		True Negative		False Negative		Accuracy	Recall	f1-score
	Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.			
Document	31,623	0.033	16,290	0.017	888,298	0.947	960	0.001	0.66	0.97	0.785
Image	367,260	0.391	1,970	0.002	540,339	0.576	27,602	0.029	0.994	0.93	0.961
Media	2,799	0.002	47	0.000	933,813	0.996	512	0.001	0.983	0.845	0.91
Stylesheet	101,775	0.108	929	0.001	833,066	0.888	1,401	0.001	0.99	0.986	0.988
Script	332,304	0.354	8,644	0.009	589,039	0.628	7,184	0.007	0.974	0.978	0.976
Font	53,385	0.056	520	0.001	872,900	0.931	10,366	0.011	0.99	0.837	0.907

Table 4.3: Content classification heuristic evaluation (total: 937,171 resources) with advanced metrics.

Validation of this heuristic implied a dataset of 25,000 websites, encompassing 937,171 resources, and involved confusion matrix construction to evaluate classification performance. Results, including precision, recall, and f1-score for each category, are detailed in Table 4.3. The f1-score, representing the harmonic mean of precision and recall, shows robust outcomes across categories, with scores exceeding 90% except for the Document category, which faced a precision challenge at 66%. This anomaly comes from instances where resources, like images, fail to load correctly (e.g., returning an `HTML` document with status code other than 200), yet are classified as images by `Chrome`'s performance tool. This discrepancy raises questions about `Chrome`'s internal decision-making processes, which are not publicly documented.

		True Positive		False Positive		True Negative		False Negative	
		Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.
Manual Inspection	EasyList	77	0.015	-	-	5,114	0.985	-	-
	EasyPrivacy	243	0.047	-	-	4,948	0.953	-	-
Complete List	EasyList	77	0.015	0	0.0	5,114	0.985	0	0.0
	EasyPrivacy	243	0.047	0	0.0	4,948	0.953	0	0.0
Pruned List	EasyList	64	0.012	0	0.0	5,127	0.988	13	0.002
	EasyPrivacy	170	0.033	0	0.0	5,118	0.986	73	0.014

Table 4.4: Advertisement heuristic validation (total: 5,191 resources).

4.4.2 Advertisement Heuristics

We can use NetQuartz's logging feature of the `URLs` of each resource (Section 3.3.4) to test them against well-known `Adblock Plus` filter lists. The goal is to detect resources related to service promotions, advertisements, or user information analysis and tracking, often conducted for the purpose of delivering personalized ads. We use two primary lists for this purpose: `EasyList` [28], focusing on advertisements, and `EasyPrivacy` [29], aimed at eliminating all forms of internet tracking–both lists have been sourced on March 25$^{\text{th}}$, 2024.

A website predominantly loads its advertisements dynamically, which requires user interaction [68] that falls outside the programmed capabilities of NetQuartz for its main features. For efficiency reasons, an additional heuristic has been implemented to reduce the large number of filter rules. The application of the latter reduced the number of rules by 39% across both lists.

We have so employed a dual approach, integrating both heuristic-based and `Adblock Plus` filtering list-based strategies. The heuristic method simplifies the selection of rule subsets to consider: general rules located at the beginning of the file are included without exception. The presence of specific keywords within a rule is then assessed to determine its eligibility for the subset, the heuristic (along with the chosen keywords) is summarized in Algorithm 1.

To validate our classification approach, we randomly selected 100 websites (encompassing a total of 5,191 resources) and initially conducted a manual inspection of advertising and tracking resources. Subsequently, we applied the full lists without any form of pruning, followed by a final pass using our pruned list approach. The results are documented in Table 4.4: we can clearly see that it is the complete lists that are more efficient regarding the manual inspection, although our approach has the particularity of exposing **no false positives**. We can therefore affirm that this heuristic shows a precise lower bound on advertising and tracking content.

Algorithm 1 Pruning heuristic for `Adblock Plus` filter lists

Input: file_path – path to the filter list file
Output: filtered_rules – list of relevant filter rules
procedure LoadRules(*file_path*)
    Initialize *filtered_rules* as an empty list
    for each line in the file at *file_path* **do**
        if line is a general_rule **then**
            Add line to *filtered_rules*
        else
            for each keyword in the list ["ad", "doubleclick", "googlesyndication", "affiliates", "banner", "track", "analytic", "pop", "taboola", "stat", "collect"] **do**
                if keyword is found in line **then**
                    Add line to *filtered_rules*
    return *filtered_rules*

Chapter 5

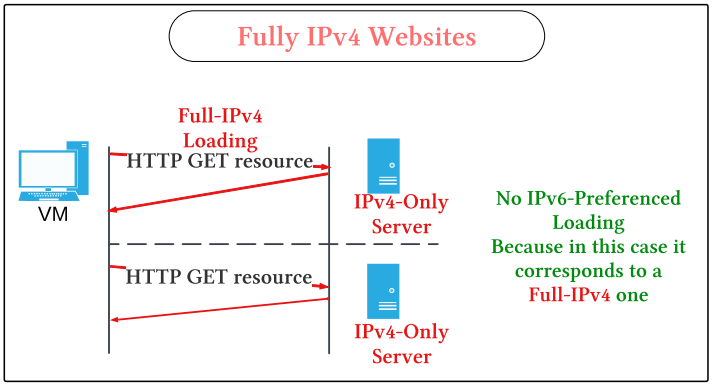
Results

The aim of this chapter is to present the main results obtained from post-processing the NetQuartz campaign carried out between March 2nd 2024 and March 25th 2024. Section 5.1 first shows the preliminary results of this campaign and the consequences for subsequent analysis, in particular, the labeling of websites into 3 different classes according to their IPv4/IPv6 loading strategy: Fully IPv4 websites, Fully Dual-Stacked websites and Mixed websites. Section 5.2 then looks at the main results relating to the content (number of HTTP/S GETs, website size, and size distribution into the different resource categories) over the 3 website classes. Section 5.3 then focuses on the largest of the classes, "Mixed " (a class forced to load a fraction of its resources in IPv4, even with IPv6-preferenced loading), to understand how much of the content is delivered over IPv4 and IPv6 when IPv6-preferenced loading occurs. The analysis will continue with a performance evaluation between the two IP address families within the three website classes (Section 5.4). Finally, we present some statistics about on user experience such as: resource failure rate analysis; advertising content and its impact on user experience; data URL resources and HTTP unsecured websites (Section 5.5).

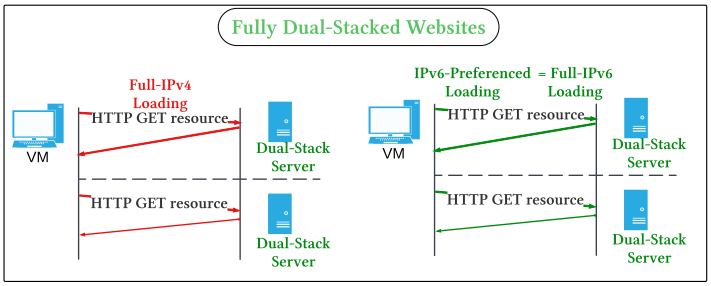
5.1 Preliminary results

Prior to the discussion of the results, we propose a classification of websites according to the way they respond to NetQuartz requests:

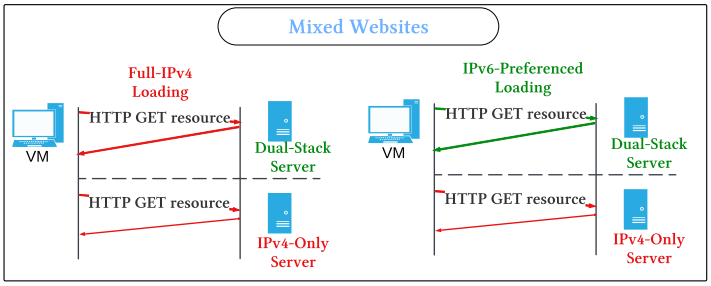
- Websites loading all their resources over IPv4-only servers can only be accessed and loaded via the IPv4 protocol. We choose to label such websites as Fully IPv4 websites.
- Websites loading all their resources over dual-stack servers are fully accessible over IPv6, but full access over IPv4 is also possible. We label such websites as Fully Dual-Stacked websites.
- We have observed websites hosting some resources on IPv4-only servers and others on dual-stack ones, supporting both IPv4 and IPv6. This setup allows those websites



(a) Class 1: `Fully IPv4` websites.



(b) Class 2: `Fully Dual-Stacked` websites.



(c) Class 3: `Mixed` websites.

Figure 5.1: Website classification summary.

to be accessed either entirely over `IPv4` or through a combination of both protocols. We label such websites as `Mixed` websites.

All website classes can be loaded using a full-`IPv4` approach (see Figure 3.2c). However, for `Fully Dual-Stacked` websites, `IPv6`-preferenced loading corresponds to a full-`IPv6` approach, as all the resources are hosted on dual-stack servers. Finally, `Mixed` websites can be accessed using an `IPv6`-preferenced loading approach, by preferring `IPv6` connections where available. Figure. 5.1 provides a summary of the different website classes along with the content loading approaches.

Table 5.1 provides an initial overview of the distribution of websites based on their class. More than 73.8% of the measured websites belong to the `Mixed` class. Both `Fully IPv4`

Mixed		Fully IPv4		Fully Dual-Stacked		No Resources	
Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.
160,920	0.738	28,861	0.132	24,454	0.112	3,738	0.017

Table 5.1: Distribution of the different website configurations (total: 217,973 websites).

and `Fully Dual-Stacked` classes are equally represented (13.2% and 11.2% respectively). Finally, Table 5.1 shows a low proportion (1.7%) of websites from which no data could be captured using `Selenium`.

5.2 Website content delivery

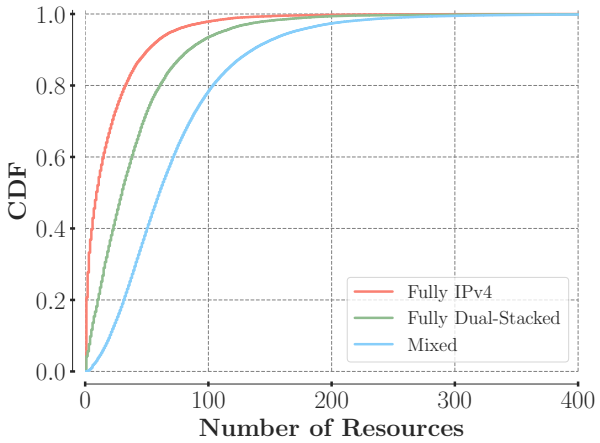
We will now look at the content delivery. In particular, we dissect the web pages collected by NetQuartz such that we expose the various *resources* composing home pages.

5.2.1 Number of Resources

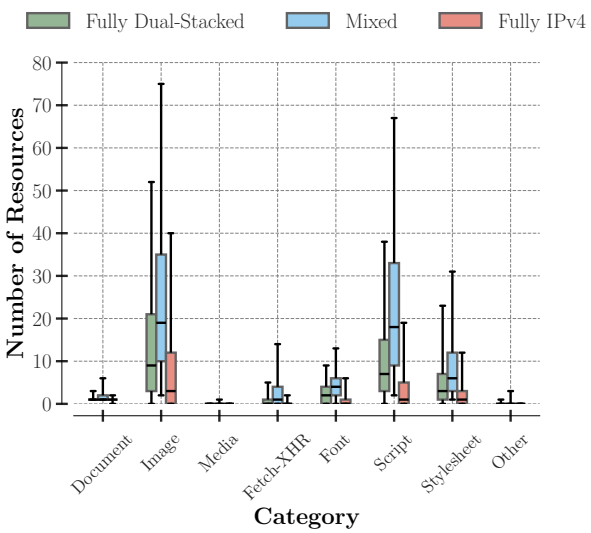
Our investigation into resource distribution directly corresponds to the number of `HTTP/S` GET requests initiated by a user. This metric is obviously independent of the `IP` protocol used for loading.

Figure 5.2a shows, as a CDF, the number of resources contained in web pages, in function of the website class. The `Fully IPv4` class consistently loads fewer resources, typically loading up to 70 resources at their 95$^{\text{th}}$ percentile. The `Mixed` class requires more resources, up to 168 at the 95$^{\text{th}}$ percentile. Finally, the `Fully Dual-Stacked` class is between the two other classes. This difference can be explained by the fact that websites tend to become larger in size and number of resources fetched over time. Considering `HTTP` Archive [39] data collected between 2012 and 2024, we find that the number of requests performed towards websites has increased by 25% in the median since 2012. As stated in Chapter 2, the `IPv6` adoption has also significantly increased over time and it is thus interesting to note that `Fully IPv4`, being older than `Fully Dual-Stacked` and `Mixed` websites, will therefore load fewer resources.

Figure 5.2b splits the downloaded resources into their respective categories, according to the website class. The distribution shows that *Image* and *Script* are the predominant categories, regardless of the website class. *Media* and *Other* categories are barely nonexistent.



(a) Total number of resources requested.



(b) Number of resources requested by category.

Figure 5.2: Number of resources requested via an `HTTP/S` GET during the loading of `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed` websites. The bottom and top whiskers represent the 5^{th} and 95^{th} percentiles, respectively. For readability reasons, outliers are not shown in the boxplots.

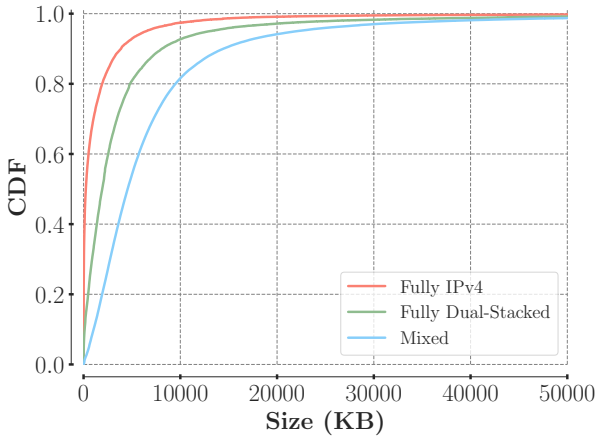
5.2.2 Website Size

Our analysis extends into the overall size of websites, noting that the total raw–meaning after decoding–data transmitted during the loading process is totally independent of whether the loading is `IPv4`-only or `IPv6`-preferenced.

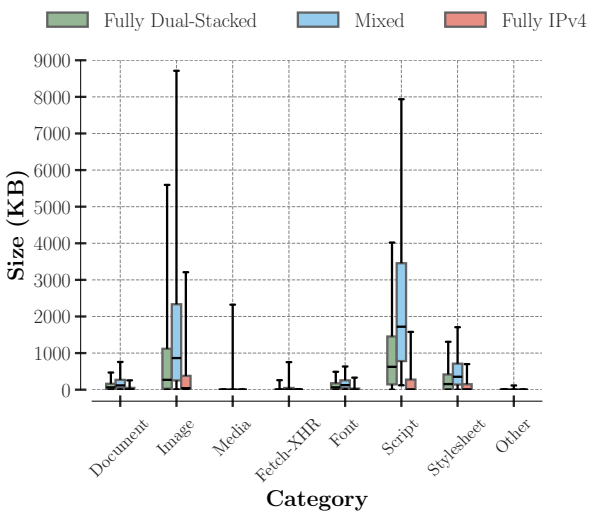
As illustrated in Figure 5.3a, `Fully Dual-Stacked` websites have the highest data size, which correlates with their higher resource numbers. Figure 5.3b shows that *Image* and *Script* are still the predominant categories, the figure also reveals a significant insight into resource utilization and efficiency, particularly in `Mixed` websites, where *Script* resources, although fewer in number, contribute disproportionately to the total data volume. This can be attributed to the complexity and functionality embedded within those scripts, which often include libraries and frameworks that are essential for modern web applications but are inherently large.

In terms of data distribution, average values provide an interesting perspective. While the majority of websites manage to maintain *Media* sizes relatively low, as indicated by the clustering of data points near the origin in the boxplot (Figure 5.3b), there are outliers with considerably large resource loads. These outliers skew the average upwards, as depicted in Figure 5.3c. This observation underscores the variable nature of web content distribution, where most websites optimize resource sizes, but some, possibly due to specific functional or design requirements, load larger *Media* files.

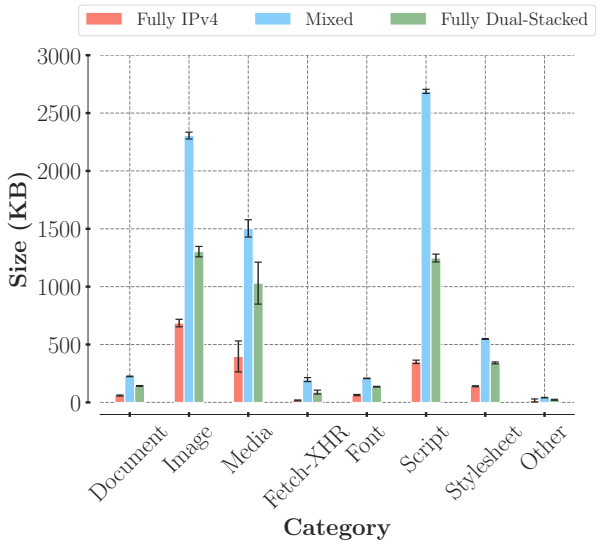
Figure 5.3d summarizes the reclassification step (see Section 4.4.1) of the resources loaded through the *Fetch & XHR* API. This is done by extracting the average size of the



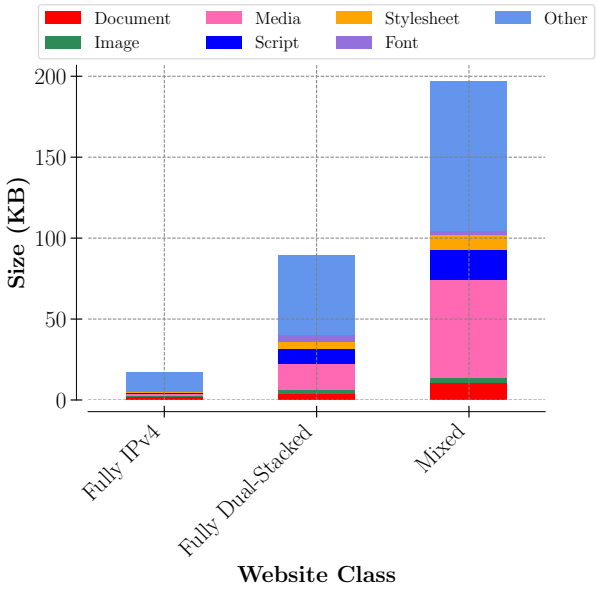
(a) Global website size.



(b) Category sizes.



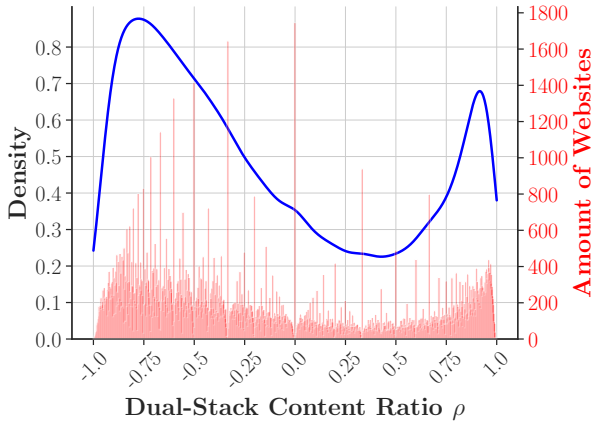
(c) Average category size with 95% confidence intervals measures.



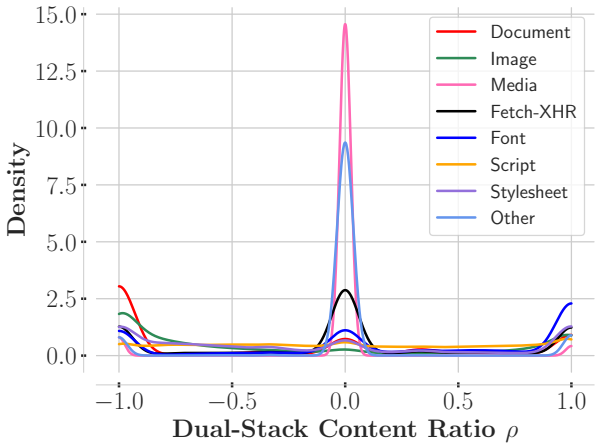
(d) *Fetch & XHR* reclassification.

Figure 5.3: `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed` websites raw size distribution (in Kilobytes). The bottom and top whiskers represent the 5^{th} and 95^{th} percentiles, respectively. For readability reasons, outliers are not shown in the boxplots.

Fetch & XHR category and subsequently illustrating how each other category contributes to this average. On average, *Media* emerges as the largest category typically loaded asynchronously. This asynchronous loading allows for enhanced user experiences by not blocking the rendering of other page elements while large media files are being loaded.



(a) General ρ.



(b) ρ calculated over each web categories.

Figure 5.4: Normalized ratio (ρ) calculated for the 162,190 `Mixed` wesbites, prioritizing more resources loaded over `IPv4` on the left-hand side (negatives values), and more resources loaded over `IPv6` on the right-hand side (positive values).

5.3 Dual-Stack content distribution

The `Mixed` websites are those which, when loaded with the default behavior of a dual-stack client (i.e. to prefer `IPv6` [62]), load a certain portion of their resources in `IPv6` and the other in `IPv4`. The aim of this section is to focus on the 160,920 so-called `Mixed` websites (see Table 5.1) and to evaluate the portion of content downloaded in `IPv6` and `IPv4` when `IPv6`-preferenced loading occurs.

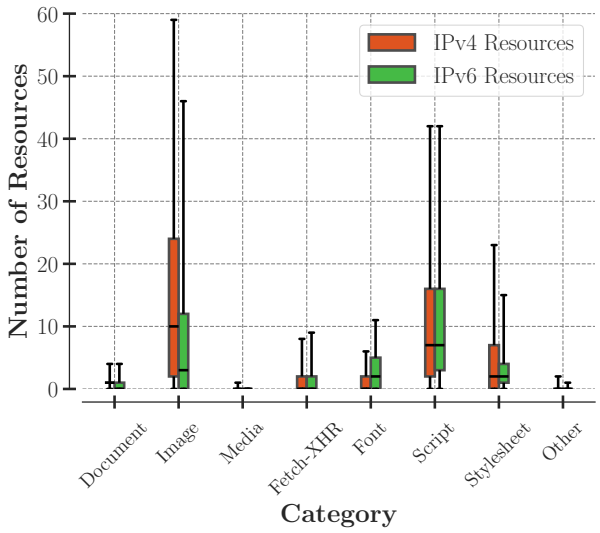
Figure 5.4a presents the probability density function of a normalized ratio calculated for each website. We express the ratio as

$$\rho = \frac{\texttt{IPv6}\_resources - \texttt{IPv4}\_resources}{total\_number}. \tag{5.1}$$

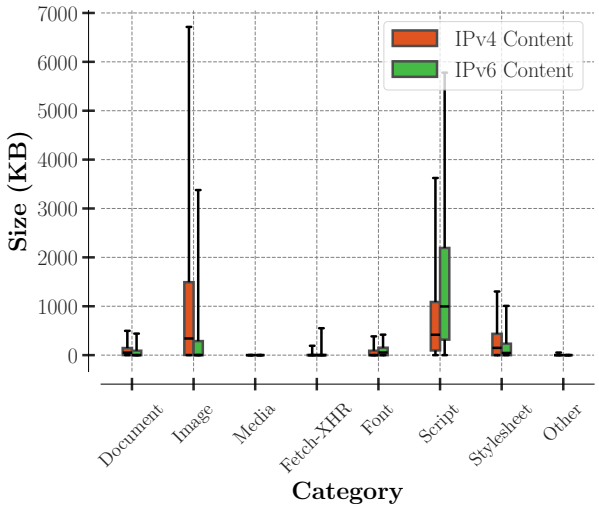
where $\rho \in]-1, 1[$. The limits of the interval (-1 and 1) are rejected as they represent the case of `Fully IPv4` and `Fully Dual-Stacked` websites respectively. A negative ρ means the website prefers to load more resources over `IPv4`, while a positive ρ means the website prefers `IPv6`. A null value means that the website equally loads resources in `IPv4` and `IPv6`.

Figure 5.4a shows that 64% of the data points fall into negative values, indicating a higher number of resources loaded over `IPv4`. Conversely, 35% of the points are positive, showcasing a preference for `IPv6`, with the remaining 1% balancing out at zero, indicating an equal distribution of resources across both protocols. The figure also includes a count of websites contributing to each bin, providing a clear view of the data distribution and the significant skew towards `IPv4`.

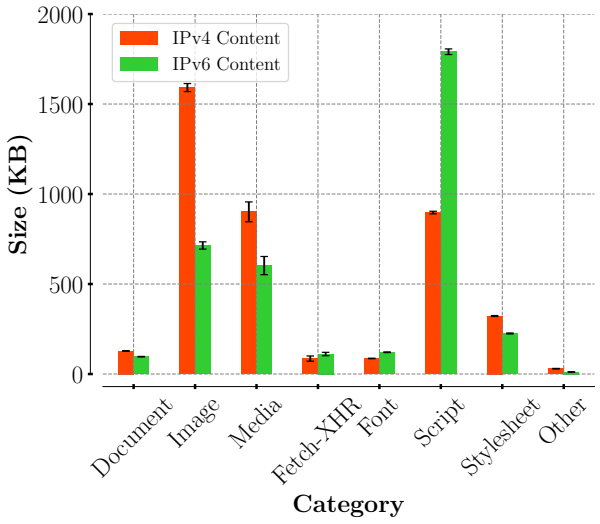
Figure 5.4b shows the same ρ calculated across the different resource categories inferred by `Selenium`. We see that for *Media*; *Other* and *Fetch & XHR* resources, the probability



(a) Distribution of resources loaded over `IPv4` and `IPv6` for the different categories.



(b) `IPv4` content size and `IPv6` content size (in Kilobytes) distribution over each category.



(c) Average portion of the size of the content loaded in `IPv4` and `IPv6` with 95% confidence intervals measures, across categories.

Figure 5.5: Dual-Stack content distribution. The bottom and top whiskers represent the 5^{th} and 95^{th} percentiles, respectively. Outliers are not shown in the boxplots.

density is concentrated around 0 according to a normal distribution. *Document* and *Image* contribute the most to the left side (giving priority to `IPv4` resources) while *Font* seems to give priority to `IPv6`. This analysis can give insights about each category's distribution, but the largest–as established in Section 5.2.1–categories remain in favor of `IPv4` resources.

Figure 5.5 delves deeper into the category of content loaded over each protocol. The box plots in Figure 5.5a demonstrate that most categories predominantly load over `IPv4`.

Image stands out the most by loading only 42% of the images over IPv6 when considering the 95th percentile. However, exceptions are observed with *Fetch & XHR* asynchronous requests and *Font*, which are more frequently loaded over IPv6. Even though the number of *Script* resources appears comparable between IPv4 and IPv6, a significant difference emerges when examining the size of these resources. Figure 5.5b reveals that scripts loaded over IPv6 tend to be larger and more complex, with sizes reaching up to 8,000 KB at the 95th percentile while reaching only the half in IPv4 content.

The analysis of average values (see Figure 5.5c) across these resource categories further enhances our understanding of protocol prioritization by showing that *Media* files also predominantly use IPv4.

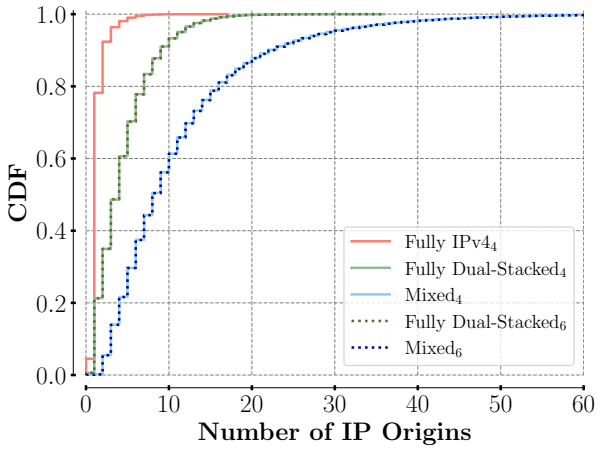
5.4 Performances

5.4.1 Parameters Influencing Performance

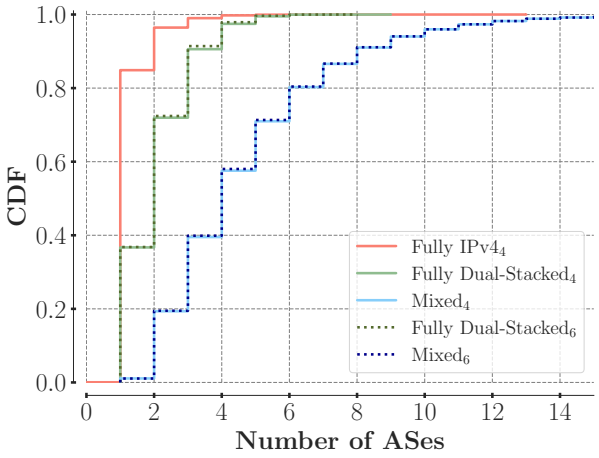
In Section 5.2 we have already considered content-related metrics such as website size and the number of resources, we now integrate performance-related metrics that potentially impact performance: the number of Autonomous Systems (ASes) and the number of IP origins involved during loading. These metrics were derived in the postprocessing phase, using MaxMind geolocation services to perform the IP to AS mapping [52].

Figure 5.6a reveals that Mixed websites tend to load resources from a larger number of different IP addresses (up to 30 at the 95th percentile), while the majority of Fully IPv4 websites ($\approx$ 80%) loads resources from maximum 2 IP addresses. Similarly, Figure 5.6b quantifies the amount of ASes involved in loading resources. Mixed websites can rely–considering the 95th percentile–up to 10 ASes, while Fully IPv4 rely on less than 2 different ASes and Fully Dual-Stacked on less than 4. It is worth noticing that, despite the different loading strategies, the global distribution of these metrics shows no significant differences. This indicates that the underlying network infrastructure treats full-IPv4 and IPv6-preferenced sessions similarly in terms of routing diversity and connection origination. We have also examined ASes relationships, relying on CAIDA dataset [12], and observed that 80% of the websites analyzed by NetQuartz hosted their main HTML document within an AS that had no direct connections to any other AS involved in the resource loading process. This suggests a significant dispersion of resources, whatever the IP protocol used. If we look deeper into these relations, we measured that–at the 95th percentile–for each website main AS, there is only up to 1 direct connection and up to 8 indirect connections to all the other ASes involved in the loading. These direct connections may be *peering* relationships or *client-supplier* relationships, but the latter are rare and we have not been able to identify any significant pattern at this level, apart from the fact that the preferred IP protocol has no influence on these parameters.

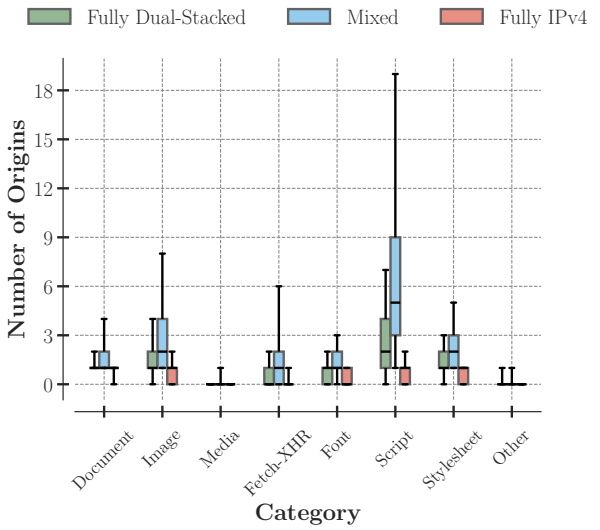
Figures 5.6c and 5.6d delve deeper into these performance-related metrics by highlight-



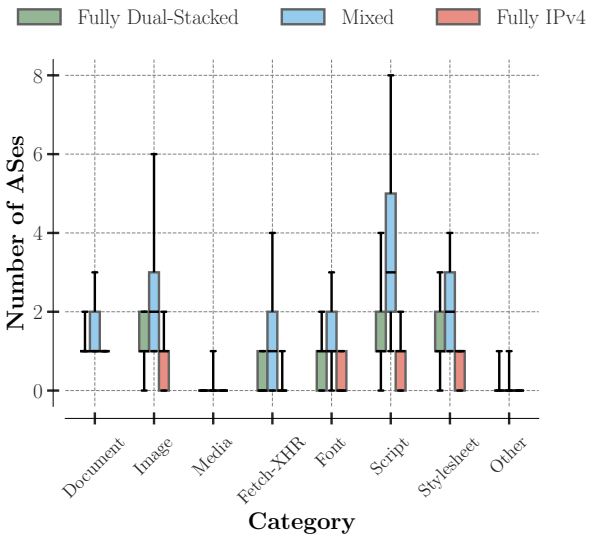
(a) Number of ≠ `IP` origins involved in the loading.



(b) Number of ≠ Autonomous Systems (`ASes`) involved in the loading.



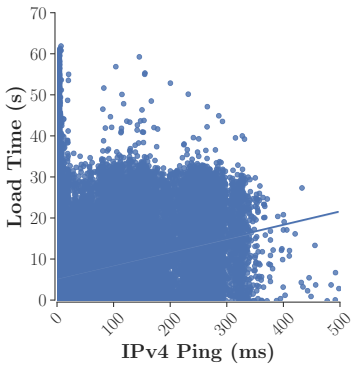
(c) Number of ≠ `IP` origins across resource categories.



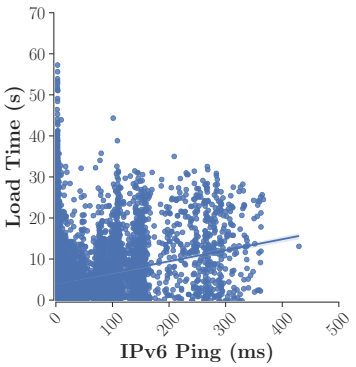
(d) Number of ≠ `ASes` across resource categories.

Figure 5.6: Performance-related metrics involved in the loading of `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed` websites. Parameters captured during `IPv6`-preferenced loading are differentiated using subscripts: subscript$_4$ showcases the use of full-`IPv4` loading while subscript$_6$ is for `IPv6`-preferenced loading. The bottom and top whiskers represent the 5$^{\text{th}}$ and 95$^{\text{th}}$ percentiles, respectively. For readability reasons, outliers are not shown in the boxplots.

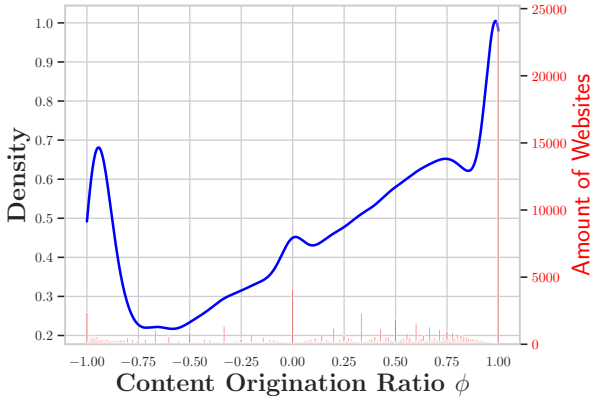
ing their distribution within each resource category. Although *Image* is one of the largest categories alongside *Script*, it is surprisingly much more balanced in terms of ≠ `IP` origins and `ASes` (whatever the website class). *Script* remains the biggest contributor above the other categories, as expected given our previous observations on the number of `HTTP/S` GET requests (see Figure 5.2b). Surprisingly, asynchronous *Fetch* & *XHR* resources are particularly present both in terms of the number of different `ASes` and `IP` origins involved



(a) Correlations between `IPv4 pings` performed on the main servers hosting the website `HTML` *Document* and the full-`IPv4` loading time of that whole website.



(b) Correlations between `IPv6 pings` performed on the main dual-stack servers hosting the website `HTML` *Document* and the `IPv6`-preferenced loading time of that whole website.

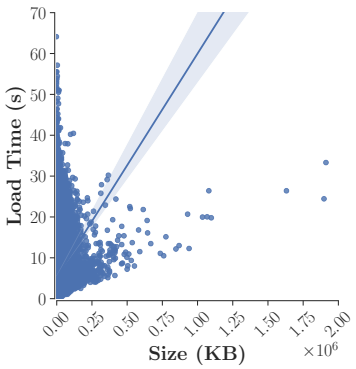


(c) Normalized ratio ϕ, prioritizing more resources loaded over a different server than the main one on the left-hand side (negatives values), and more resources loaded over the same main server on the right-hand side (positive values).

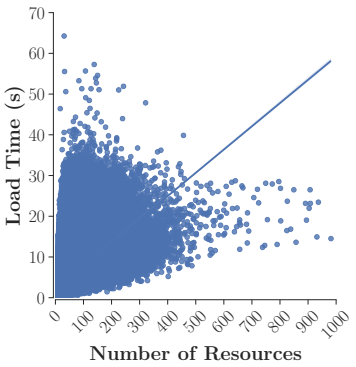
Figure 5.7: `RTT` estimations correlations results.

in their loading. This indicates that the few resources loaded asynchronously are likely to be loaded from different origins, whether at the scale of an `AS` or an `IP` address.

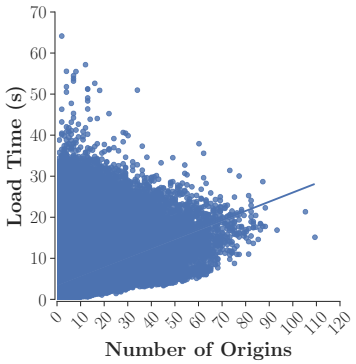
To determine what can influence website load times, we first performed a correlation study between `RTT pings` estimations logged by NetQuartz (see Section 3.3.2) and website load times. We selected and studied both websites having an `IPv4`-only main `HTML` server and a dual-stack one. `IPv4 pings` (see Figure 5.7a) were compared over both `IPv4`-only and dual-stack servers hosting the main `HTML` while, for `IPv6 pings` (see Figure 5.7b), we were logically forced to consider the subset of dual-stack main `HTML` servers. The figures show poor correlations, the first explanation being that the scale of a single server is not



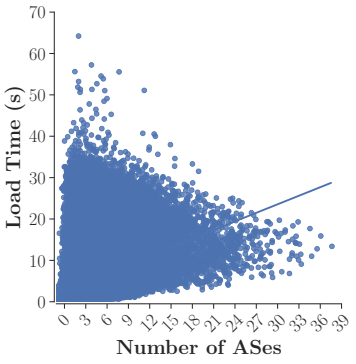
(a) Website size.



(b) Number of `HTTP/S` GET requests.



(c) Number of $\neq$ `IP` origins involved in the loading.



(d) Number of $\neq$ `ASes` involved in the loading.

Figure 5.8: Correlations performed on both content-related and performance-related metrics. `IPv6`-preferenced loading (which reduces to full-`IPv4` loading in the case of `Fully IPv4` websites) has been considered for the whole analysis.

applicable to an entire website. To prove this, we are now considering a ratio similar to the one presented in Section 5.3, but computed as follows:

$$\phi = \frac{same\_origin\_resources - different\_origin\_resources}{total\_number}. \tag{5.2}$$

where $\phi \in [-1, 1]$. The limits of the interval are such that: **1** represents the case where all resources are loaded from the main `HTML` hosting server and **-1** is the case such that all the resources are loaded from a different server than the main one. A negative ϕ means the website prefers to load more resources over servers different than the one hosting the main `HTML`, while a positive ϕ means the website prefers content from the same origin. A null value means that the website equally loads resources from the same origin and $\neq$ origins. Figure 5.7c shows the distribution of these ratios over all the websites, 35% of the data points fall into negative values; 63% fall into positive ones; 2% are at 0. We should also note that a large number of sites (around 10%) are at 1, pushing the probability density

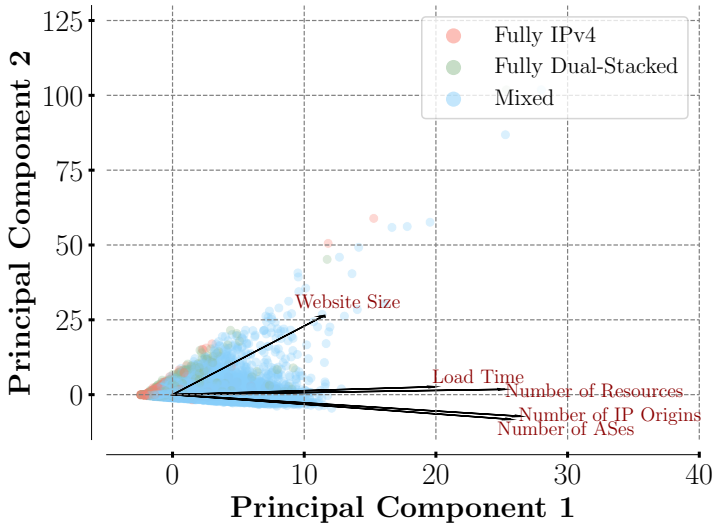


Figure 5.9: PCA biplot visualizing principal contributions of performance-related metrics such as website size, load time, number of resources; IP origins; ASes- The 2 Principal Components explain 76% of the variance (56% explained by PC1 and 20% explained by PC2). IPv6-preferenced loading (which reduces to full-IPv4 loading in the case of Fully IPv4 websites) has been considered for the whole analysis.

to the right. Taking this into account, the probability–for the remaining 90%–of having servers different from the main one involved in loading is considerable. We would also like to point out here that basing ourselves on the scale of a single server, as in previous studies linked to this subject, is not completely representative of a website as perceived by a user. This is one of the first gaps that our work aims to identify and correct.

To explain load times, we can also perform individual correlations on each of the metrics (content-related and performance-related) that we have at our disposal, the results of which are shown in Figure 5.8. The correlations for the size of the website and the number of ASes are particularly poor. A qualitative study enabled us to demonstrate that e.g. certain websites with a large size sometimes load faster because they require a smaller number of HTTP/S GET requests. Conversely, some websites generating very few requests load too slowly because they actually have a relatively large total size. This demonstrates that the problem cannot in fact be reduced to a 2D problem when the entire complexity of a website is considered, the approach must be multiplexed. This is why, to determine how these metrics interrelate and contribute to website load times, we performed a two-dimensional Principal Component Analysis (PCA). This dimensionality reduction technique helps us visualize the correlations among the variables. The results, depicted in Figure. 5.9, illustrate that vectors (arrows) with similar orientations are likely to be correlated. We note that the number of resources requested over HTTP/S significantly correlates with load times. Other features such as website size, the number of ASes, and IP origins do not show strong individual correlations with load times. However, when

considered together (size + ASes + IP origins), their cumulative impact becomes apparent. These observations suggest that a dual view of both content-related and network-centric metrics is essential to fully understand website performance dynamics.

5.4.2 IPv4 vs. IPv6 website load time

It has been established in Section 5.4.1 that the number of resources significantly impacts load times, with Fully IPv4 websites loading fewer resources compared to Fully Dual-Stacked, and subsequently, Mixed websites. The same relationship can be observed by considering the loading time of the website (Figure 5.10a), classes loading fewer resources will therefore load faster.

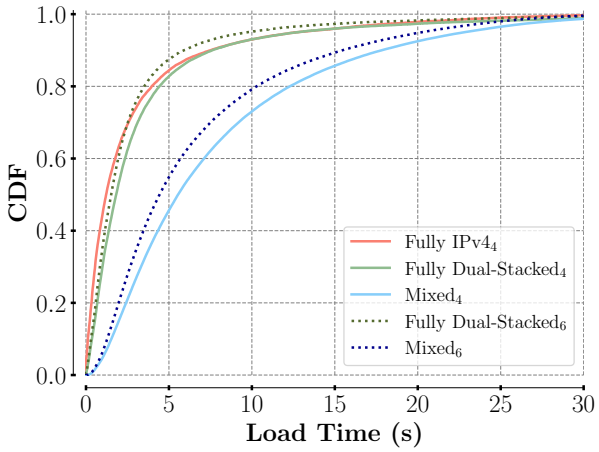
Regardless of the website's class, IPv6-preferenced sessions outperform the full-IPv4 ones. Mixed class loads 20% faster in median and 11% faster at their 95$^{\text{th}}$ percentile when using IPv6-preferenced loading. The difference is even more noticeable in the case of Fully Dual-Stacked class, IPv6-preferenced load 16% faster in median and 24% faster at the 95$^{\text{th}}$ percentile. This observation is further pointed out by the boxplots in Figures 5.10b, 5.10c, 5.10d showing the distribution of loading times across the various content categories. Predominantly, *Image* and *Script* remain the dominant categories, yet in all instances, IPv6-preferenced loading times are slightly better than those of full-IPv4.

Typical justifications for the enhanced performance of IPv6 include the absence of NAT processing, reduced header processing at network nodes, and the avoidance of fragmentation [44, 20]. However, in our case, the argument regarding NAT processing does not hold since our vantage points use publicly addressable IP addresses.

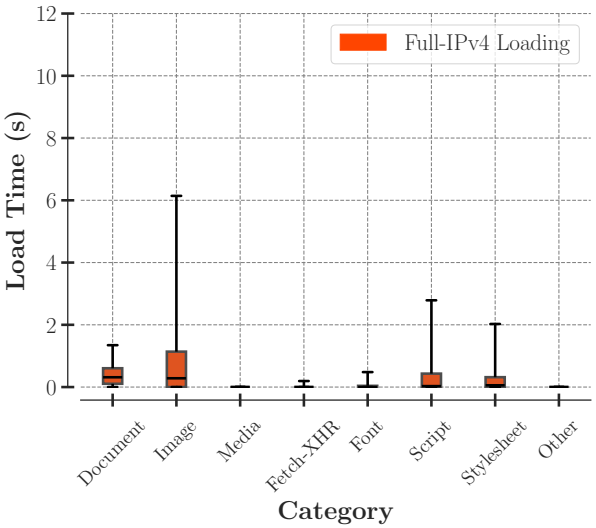
To investigate deeper into potential reasons behind IPv6's performance advantage, we used specific data provided by NetQuartz: the average Round Trip Time (RTT) extracted from the five pings performed to the server hosting the main HTML document (see Section 3.3.2) and the IP addresses logged for each resource during the loading process (see Section 3.3.4).

We first examined the RTT estimates, Figure 5.11 shows that the distributions between IPv4 and IPv6 for dual-stack servers are the same, so RTT estimation is not an ideal candidate to explain this difference in performance. Nevertheless, there is a clear difference in performance between IPv4-only servers and dual-stack servers, with pings towards dual-stack servers responding 78% faster for the 95$^{\text{th}}$ percentile and 71% faster in the median.

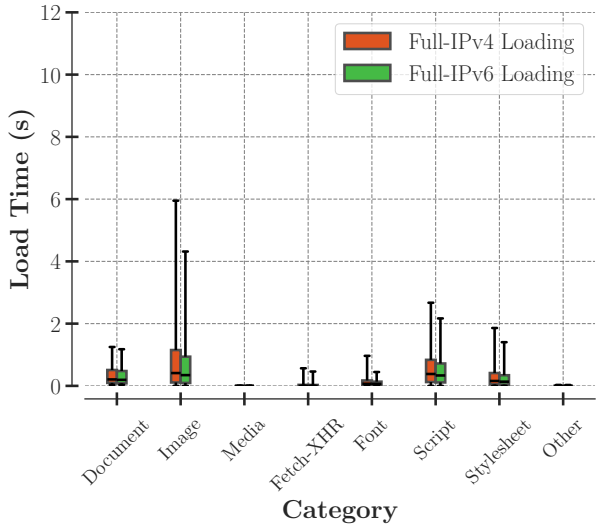
We then calculated geographical distance–through MaxMind geolocation services [52]–using the Haversine formula to measure the distance between each server hosting a resource and the corresponding vantage point. These distances were weighted by the resource size, scaled down by a factor of 10^6 for readability reasons, producing what we refer to as the "HaverSize score". Assuming we have n resources, each with a resource size s_i (in Kilobytes) and a Haversine distance d_i from the vantage point (in kilometers), and K is



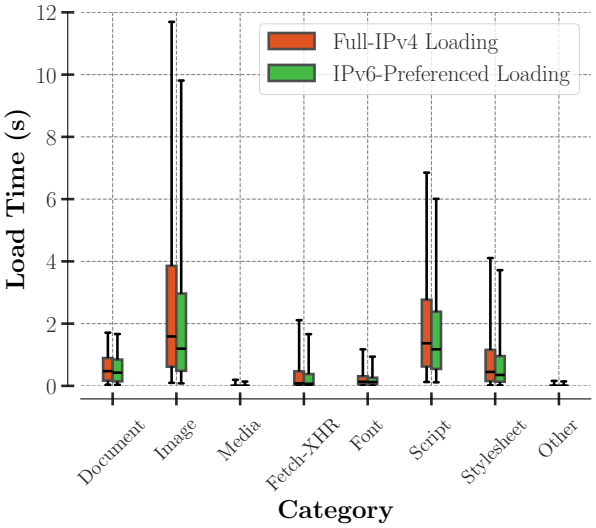
(a) Global load time for `Fully IPv4`, `Fully Dual-Stacked` and `Mixed` websites.



(b) Loading time distribution for `Fully IPv4` class.



(c) Loading time distribution for `Fully Dual-Stacked` class.



(d) Loading time distribution for `Mixed` class.

Figure 5.10: Website load time distribution (in seconds) of `Fully IPv4`, `Fully Dual-Stacked` and `Mixed` websites. Subscript$_4$ showcases the use of full-`IPv4` loading while subscript$_6$ is for `IPv6`-preferenced loading. The bottom and top whiskers represent the 5$^{\text{th}}$ and 95$^{\text{th}}$ percentiles, respectively. Outliers are not shown in the boxplots.

the scaling factor set to 10^6, then the `HaverSize` score of a website is calculated as follow:

$$\texttt{HaverSize} = \frac{\sum_{i=1}^{n} s_i \times d_i}{K}. \tag{5.3}$$

The `HaverSize` score is a non-negative metric (≥ 0), a score close to zero indicates that the sum of the weighted distances for all resources is minimal. Practically, this suggests that most resources are either very small in size or are close to the vantage point.

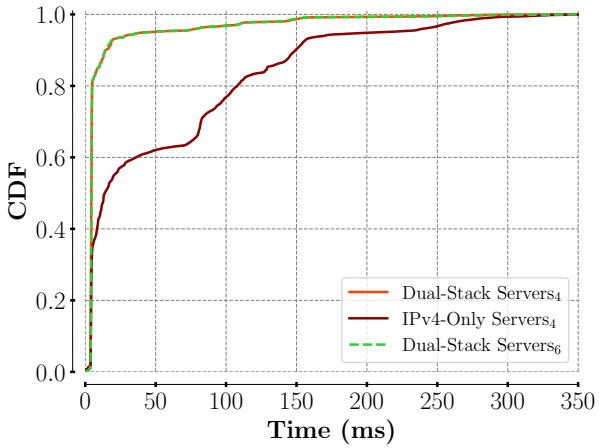


Figure 5.11: `ping` response times (in milliseconds) for `IPv4` vs. `IPv6` on dual-stacked and `IPv4`-only classes. Subscript$_4$ means that the `ping` is performed over `IPv4` loading while subscript$_6$ is for `IPv6` `pings`.

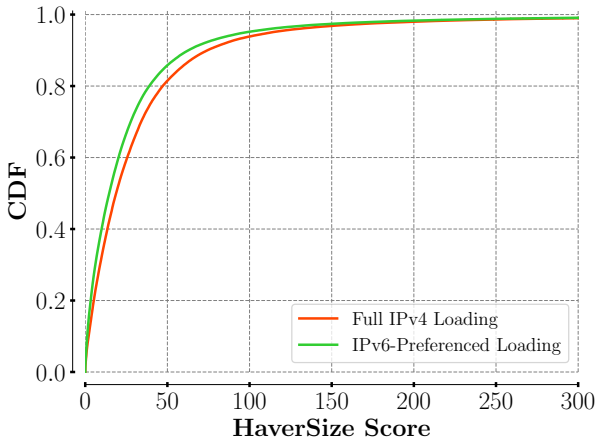
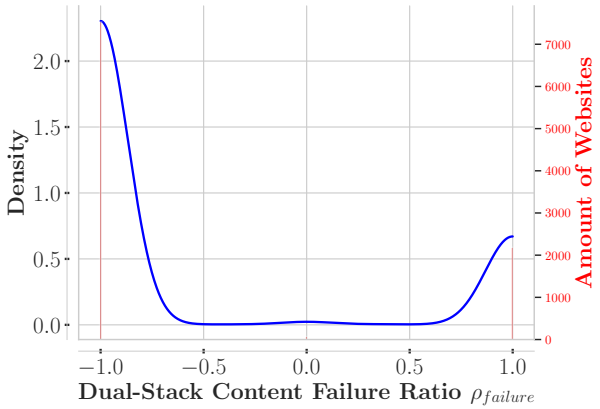


Figure 5.12: HaverSize scores, where each score is the weighted sum of Haversine distance (in kilometers) between the vantage point and the resource server, with weights proportional to resource sizes, comparing full-`IPv4` and `IPv6`-preferenced loading strategies.

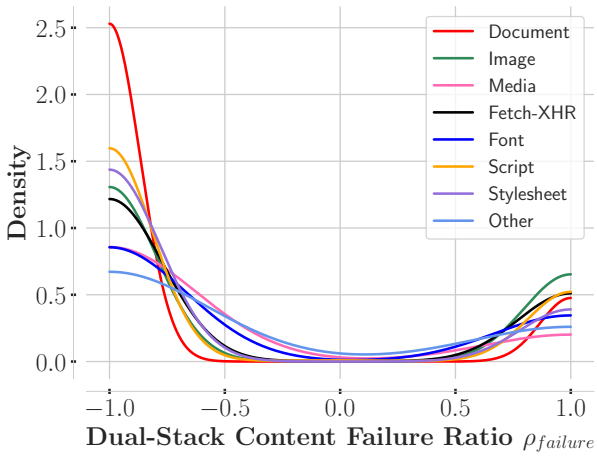
Figure 5.12 highlights the distribution of these `HaverSize` scores, which consistently show a preference for `IPv6`. This suggests that `IPv6`-preferenced loading leads to routes that might be more efficient or involve shorter physical distances. It is important to note that `IP` geolocation and Haversine calculations do not precisely represent the exact network paths taken by `IP` packets, they still provide useful insights into the underlying website infrastructure and the physical proximity of connected resources.

Failure Type	Full-IPv4 Session		IPv6-Preferenced Session	
	Raw	Prop.	Raw	Prop.
Read Timeout	11,850	0.0553	10,092	0.047
Connect Timeout	2,269	0.01	2,048	0.01

Table 5.2: Failure rate table showing the raw number and the proportion of `Mixed` websites that have experienced at least one failure (either during a connection or a reading attempt and either during the full-`IPv4` session or `IPv6`-preferenced one).



(a) General $\rho_{failure}$.



(b) $\rho_{failure}$ distributed across resource categories.

Figure 5.13: Normalized ratio ($\rho_{failure}$) calculated for the `Mixed` wesbites, prioritizing more resources that have failed over `IPv4` on the left-hand side (negatives values), and more resources that have failed over `IPv6` on the right-hand side (positive values).

5.5 A step into user experience

5.5.1 Resource failure

Failed resources can compromise the display of a website, and therefore the user experience, if they fail to download. In this section, we will explore failure rates at the transport layer, while failures at the application layer (detectable through `HTTP` error status codes) will not be explored within the scope of this work. We will also try to see whether these failure rates differ according to the `IP` protocol used to fetch the resource. Table 5.2 shows the rate of websites having experienced at least one failure of a certain type. This table gives us some initial insights, showing that *connection failures* are much less common than *read failures*. Directly comparing the full-`IPv4` session with the `IPv6`-preferenced session still does not give us enough information as to which `IP` protocol is the source of the failure, since, as a

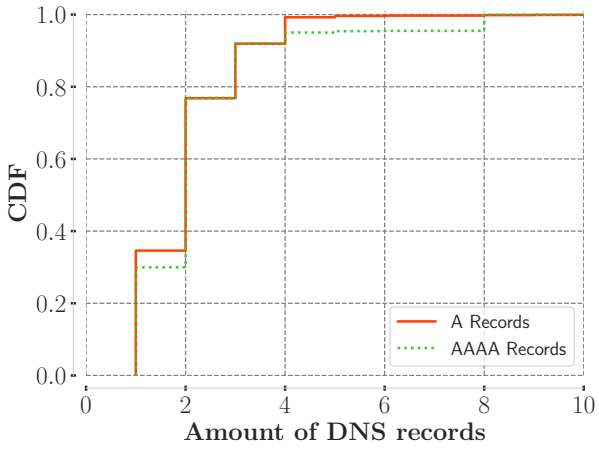


Figure 5.14: Distribution of the number of `A` and `AAAA` `DNS` records associated to the server hosting the main `HTML` document of the website.

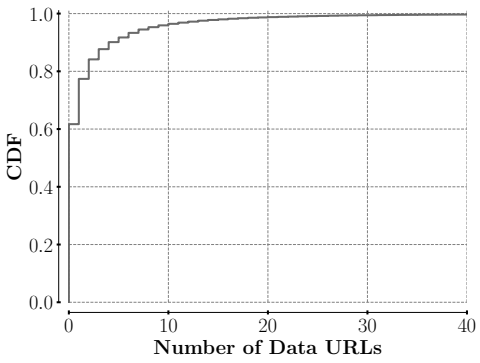
reminder, `IPv6`-preferenced loading does not exclude the use of `IPv4` (see Section 5.1).

To solve this, let us consider the most prominent website class, `Mixed`, and re-consider a modified version of the ratio ρ introduced in Section 5.3. Our new ratio is $\rho_{failure}$ and is constructed by normalizing the difference between the number of resources that have failed (either *read timeout* or *connect timeout*) over `IPv6` and `IPv4`, i.e:

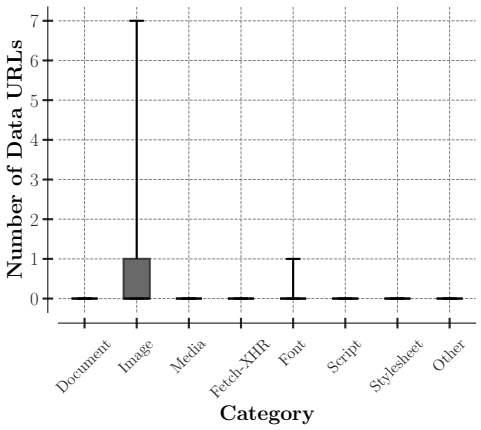
$$\rho_{failure} = \frac{\texttt{IPv6}\_failed\_resources - \texttt{IPv4}\_failed\_resources}{total\_number}. \tag{5.4}$$

Let us now consider the distribution (Figure 5.13) of this ratio applied to `Mixed` websites that have experienced at least one failure (9,842 websites in total). Figure 5.13a highlights that most values fall into 1 or -1, with a preference for -1 (indicating a higher number of failures over `IPv4`). So it seems that when a website experiments failure, it will either experiment it completely over `IPv4` or completely over `IPv6`: 76% of the data points fall into -1 value and 22% fall into 1. However, we should also remember Figure 5.4a, which showed that `IPv4` resources are predominant by default, which makes our previous observation even more relevant.

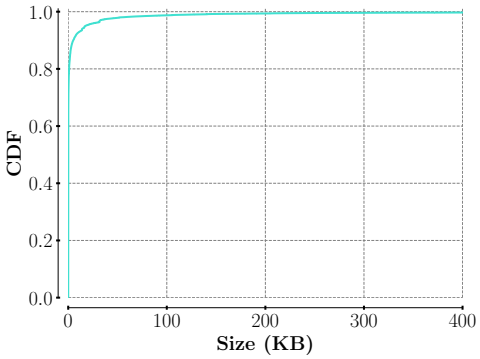
Figure 5.13b extends the analysis to the different resource categories, showing that *Document* are the most prone to such failures. If the *Document* in question is the main `HTML`, this can become problematic. That said, NetQuartz has been designed to test only the first `IP` address returned by getaddrinfo() when it fetches the resource, whereas typically, a failure would involve testing the next `IP` addresses. Hopefully, NetQuartz gives us the opportunity to see how many backup solutions, in terms of `DNS` records, the main `HTML` hosting server offers (see Section 3.2b). These are shown in Figure 5.14: in both `A` and `AAAA` records, all websites offer up to 8 backups at their 95$^{\text{th}}$ percentile.



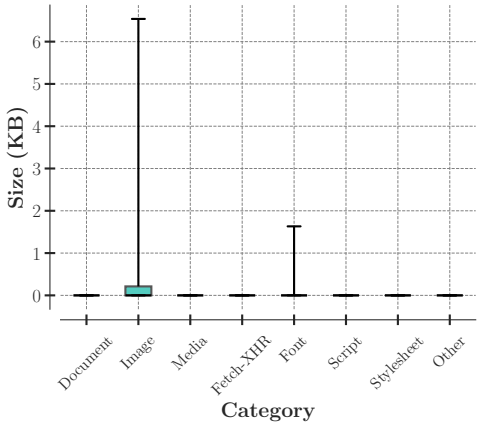
(a) Distribution of the number of `data URLs`.



(b) Distribution of the number of `data URLs` across resource categories.



(c) Distribution of the size (in Kilobytes) of `data URLs` across resource categories.

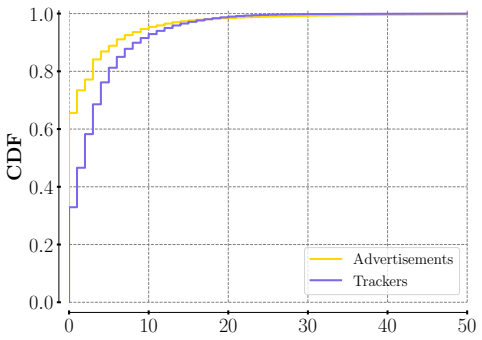


(d) Distribution of the size (in Kilobytes) of `data URLs` across resource categories.

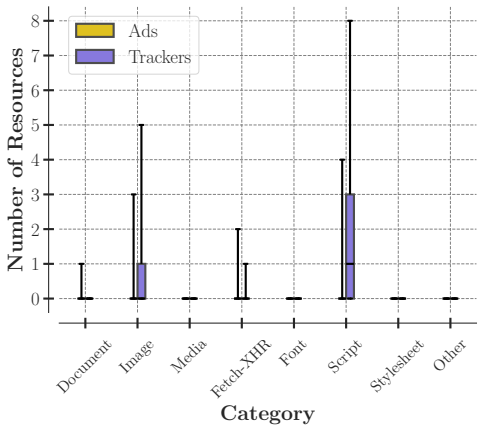
Figure 5.15: `data URLs` distribution. The bottom and top whiskers represent the 5th and 95th percentiles, respectively. Outliers are not shown in the boxplots.

5.5.2 `data URL` resources

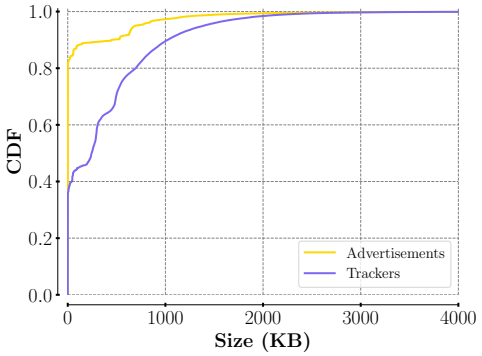
When it comes to user experience, the most important parameter is certainly website loading time. As established in Section 5.4.1, the number of resources requested via `HTTP/S` significantly influences load time. In light of this observation, many websites are adopting the use of `data URLs` to embed data directly within *Document* resources (NetQuartz Step 3 – see Section 3.3.3). This approach helps to minimize `HTTP` overhead by reducing the number of separate requests required for resource fetching. We examined the usage of `data URLs` across various websites and found that 38% of them incorporate this technology. At the 95th percentile, Figure 5.15a shows that each website contains up to 8 `data URLs`, embedding a total of 17.2 Kilobyte of data within these `URLs` (Figure 5.15c). We have also observed that the majority of these `data URLs` are concentrated in the *Image* category, with



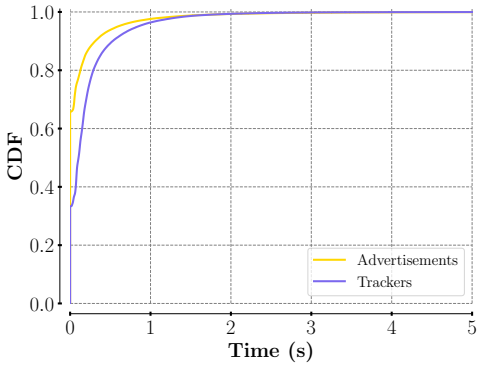
(a) Number of advertising and tracking resources.



(b) Number of ads and trackers across resource categories.



(c) Size (in Kilobytes) of advertising and tracking resources.



(d) Time spent loading advertising and tracking resources.

Figure 5.16: Ads and trackers distribution. The bottom and top whiskers represent the 5$^{\text{th}}$ and 95$^{\text{th}}$ percentiles, respectively. Outliers are not shown in the boxplots.

up to 7 `data URLs` *Image* resources and up to 1 in the *Font* category (see Figure 5.15b), the other categories are almost non-existent. We can therefore see that most websites introduce this technology to load small images, mainly saving several `HTTP/S` fetches for these small resources, at the cost of a slightly heavier `HTML` document in terms of Kilobytes.

5.5.3 Advertising

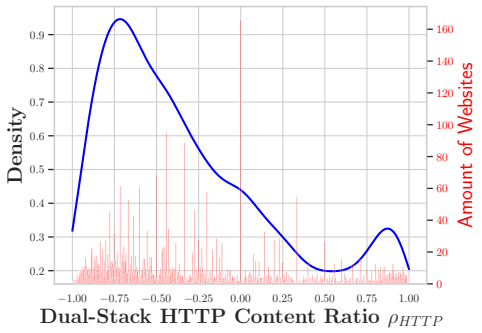
We cannot talk about user experience without introducing advertising. Most websites use this approach to sell their service or third-party services in return for payment. NetQuartz allows us to establish a lower bound on these ads (see Section 4.4.2). We are interested in ads as such but also in tracking resources which are particularly useful for collecting information about the user and delivering personalized advertising accordingly. Figure 5.16a shows that, at the median, we have no ads and up to 2 trackers, while at

the 95th percentile, we have up to 10 ads and 12 trackers. Figures 5.16d and 5.16c delve deeper into user experience by showing that, at the 95$^{\text{th}}$ percentile, users spend up to 0.6 seconds loading ads (encompassing 693 KB) and 0.8 seconds loading trackers ($\sim$ 1400KB). Furthermore, Figure 5.16b highlights that these ads and trackers are mainly within the *Script* categories, it is thus not just about fetching them, it is about executing them–which adds an extra delay. We are just trying to raise the ecological point here, by asking whether it is really necessary to devote such a large amount of network resources for the sole purpose of promoting services. Remember that NetQuartz only detects a lower bound on these ads, and NetQuartz has not been configured to capture all dynamic resources (see Section 4.4.2) (in particular by manually scrolling down the page), which also explains why *Fetch & XHR* is not the dominant category in Figure 5.16b.

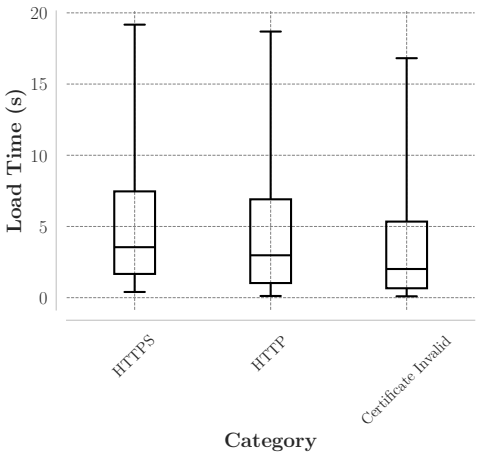
5.5.4 Security

Our aim here is to open a parenthesis on the security aspect of websites, taking advantage of NetQuartz's pass into the `DomCop`'s top300,000 domain list to take a look at the issue of website security in 2024. NetQuartz has fetched the certificate of each main service domain (see Section 3.3.1) and we observed that 2 % of the websites analyzed provided an expired certificate at the time of fetching, i.e. 4,479 websites. We also logged and analyzed all the websites that only accepted `HTTP` connections, expressly refusing or ignoring the `HTTPS` request on their main server (the one hosting the main `HTML`), i.e. 8874 of them (or 4%). A special feature of these `HTTP` websites is that they load almost all their resources over `IPv4`, Figure 5.17a shows the distribution of ρ_{HTTP} which is the same ρ ratio already introduced in Section 5.3 but applied to `HTTP` websites only. The preference for `IPv4` is more pronounced in this context, with 77% percent of sites falling into negative values vs. 64% percent for normal rho (recall from Figure 5.4a). We therefore conclude that `HTTP` websites tend to load slightly more over `IPv4` (and less over `IPv6`) than normal websites.

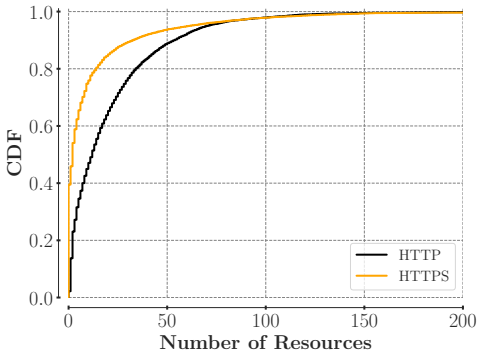
So far, our definition of an `HTTP` website has been limited to the fact that the main server has expressly refused or ignored our `HTTPS` requests, but what about the other servers that make up the website? More specifically, it is important to note that displaying an `HTTP` website involves downloading not just one, but a multitude of resources over `HTTP`–see Figure 5.17c. This makes access to the website even more insecure. Conversely, a certain number of resources of `HTTP` websites are still downloaded over `HTTPS` (they mainly come from other origins e.g. from google servers refusing `HTTP` connection).



(a) Normalized ratio (ρ_{HTTP}) calculated for the `HTTP` wesbites, prioritizing more resources loaded over `IPv4` on the left-hand side (negatives values), and more resources loaded over `IPv6` on the right-hand side (positive values).



(b) `IPv6`-preferenced loading times of: `HTTPS` websites, `HTTP` websites and websites that have delivered an expired certificate.



(c) Distribution of the number of `HTTP` and `HTTPS` resources loaded within `HTTP` websites.

Figure 5.17: Unsecured `HTTP` websites analysis.

Chapter 6

Conclusion

This thesis contributes to understanding dual-stack server performance through the development and deployment of NetQuartz, a website data analyzer tool. NetQuartz is designed to derive and compare the performance of websites across both `IPv4` and `IPv6` protocols. We successfully collected and analyzed data from more than 200,000 websites after a two-week measurement campaign.

Our analysis categorizes websites into three distinct classes based on the servers hosting their resources: `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed`. This classification allows us to uncover patterns in web resource management and loading efficiencies. Notably, `Fully IPv4` class are generally smaller in size compared to `Fully Dual-Stacked` class, with `Mixed` class presenting the largest sizes and the greatest number of resources. This observation is particularly insightful as it correlates with our finding that the number of resources is directly linked to loading times.

Our findings also reveal that `Fully Dual-Stacked` and `Mixed` classes typically perform better in `IPv6`-preferenced loading than in full-`IPv4` one.

The observation is particularly interesting with `Mixed` websites, where we find that resources tend to be loaded more over `IPv4` than `IPv6`, yet the overall performance in `IPv6`-preferenced loading remains more efficient.

6.1 Future works

Our findings could benefit, in the same way as similar work [42, 7], from a longitudinal study in which NetQuartz would make well-defined and periodic passes over an identified subset of the `DomCop` list. In addition, we limited ourselves to three measurement points during this campaign, although well distributed geographically, the *European* machine was allocated almost the entire initial sample i.e. $\sim 82\%$, *Asia* and *North America* each gets $\sim 9\%$ of the latter. This implies that repeating the study on a large number of measurement points could shed light on the increased efficiency of `IPv6`-preferenced loading, thus discarding the hypothesis of having carried out the measurements in an

overly "IPv6-friendly" environment. Finally, integrating `traceroute` measurements within NetQuartz could give a better estimate of the distance, in terms of number of hops, between the measurement point and the target servers. This would require an efficient methodology and tuning so as not to compromise NetQuartz's computing speed.

6.2 Ethical considerations

The collection of web resources can be a contentious topic, often raising ethical concerns and attracting attention. In-depth analysis of web content is generally considered unacceptable, and many web services and hosts have anti-scraping and anti-bot measures in place to discourage such activity. It is important to note that the focus of this project has been on collecting metadata related to resources, such as size, loading time, or resource category, rather than harvesting their content.

Additionally, we have intentionally avoided using techniques to bypass anti-scraping measures to ensure that no web service is deceived or impersonated, whether by simulating a human user or a web browser. Furthermore, all heuristics used are exclusively based on information that is either readily available or explicitly provided to the user.

This thesis aims to provide valuable insights into web resource dynamics while adhering to ethical guidelines and focusing on metadata. The integrity and security of web entities were not compromised.

Bibliography

[1] 2024. *MIME Sniffing Standard.* Living Standard. WHATWG: Web Hypertext Application Technology Working Group. `https://mimesniff.spec.whatwg.org/` [Last Accessed: February 19th, 2024].

[2] F. Abel, E. Herder, G.-J. Houben, N. Henze, and D. Krause. 2013. Cross-system User Modeling and Personalization on the Social Web. *User Modeling and User-Adapted Interaction* 23 (2013), 169–209.

[3] V. Bajpai. 2016. simweb, A tool to measure similarity of webpages delivered over IPv4 and IPv6. `https://github.com/steffiejacob/simweb` [Last Accessed: March 28th, 2024].

[4] V. Bajpai, J. Ott, and J. Schonwalder. 2015. Measuring Youtube from Dual-Stack Hosts. In *Proc. Passive and Active Measurement Conference (PAM).*

[5] V. Bajpai and J. Schönwälder. 2013. Measuring TCP Connection Establishment Times of Dual-Stacked Web Services. In *Proc. Conference on Network and Service Management (CNSM).*

[6] V. Bajpai and J. Schönwälder. 2016. Measuring the effects of happy eyeballs. In *Proc. ACM/IRTF Applied Networking Research Workshop (ANRW).*

[7] V. Bajpai and J. Schönwälder. 2019. A Longitudinal View of Dual-stacked Websites – Failures, Latency and Happy Eyeballs. *IEEE/ACM Transactions on Networking (ToN)* 27, 2 (April 2019), 577–590.

[8] V. Bajpai and J. Schönwälder. 2019. A Longitudinal View of Dual-stacked Websites - Failures, Latency and Happy Eyeballs (Software and Dataset). `https://github.com/vbajpai/2019-ton-v6-websites-analysis`. (2019). [Last Accessed: February 24th, 2024].

[9] C. Benfield, N. Prewitt, and Contributors. 2024. requests python library. `https://pypi.org/project/requests/` [Last Accessed: May 24th, 2024].

[10] F. Brockners, S. Bhandari, and T. Mizrahi. 2022. *Data Fields for In-Situ Operations, Administration, and Maintenance (IOAM).* RFC 9197. Internet Engineering Task Force.

[11] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. 2011. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proc. ACM Internet Measurement Conference (IMC).*

[12] CAIDA. 2024. *AS Relationships.* `https://www.caida.org/catalog/datasets/as-relationships/` [Last Accessed: April, 18th 2024].

[13] B. Carpenter and S. Jiang. 2013. *Transmission and Processing of IPv6 Extension Headers.* RFC 7045. Internet Engineering Task Force.

[14] B. Carpenter, M. Nottingham, and K. Moore. 2001. *Connection of IPv6 Domains via IPv4 Clouds.* RFC 3056. Internet Engineering Task Force.

[15] RIPE Network Coordination Center. 2010. Atlas. `https://atlas.ripe.net` [Last Accessed: February 18th, 2024].

[16] Chrome. 2024. DevTools: Chrome For Developers. `https://developer.chrome.com/docs/devtools?hl=en` [Last Accessed: May 24th, 2024].

[17] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. 2010. Evaluating IPv6 adoption in the Internet. In *Proc. Passive and Active Measurement Conference (PAM).*

[18] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey. 2014. Measuring IPv6 adoption. In *Proc. ACM SIGCOMM.*

[19] DARPA. 1981. *Internet Protocol.* RFC 791. Internet Engineering Task Force.

[20] J. Davies. 2012. *Understanding IPv6* (3rd ed.). Microsoft Press. 91–115 pages. [Chapter 4: The IPv6 Header].

[21] S. Deering and R. Hinden. 1998. *Internet Protocol, Version 6 (IPv6).* RFC 2460. Internet Engineering Task Force.

[22] S. Deering and R. Hinden. 2017. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200. Internet Engineering Task Force.

[23] A. Dhamdhere, M. Luckie, B. Huffaker, kc claffy, A. Elmokashfi, and E. Aben. 2012. Measuring the deployment of IPv6: topology, routing and performance. In *Proc. ACM Internet Measurement Conference (IMC).*

[24] DigitalOcean. 2024. Cloud Infrastructure for Developers. `https://www.digitalocean.com` [Last Accessed: April 28th, 2024].

[25] Chromium Distribution. 2024. Webdrivers for Chrome (download page). `https://chromedriver.chromium.org/downloads`.

[26] DomCop. 2023. Top 10 million Websites Based on Open PageRank data. `https://www.domcop.com/top-10-million-websites` [Last Accessed: December 19th, 2023].

[27] D. Eastlake, Smith C., and D. Soroka. 2000. *HTTP MIME Type Handler Detection.* RFC 2936. Internet Engineering Task Force.

[28] EasyList. 2024. EasyList - advertizement filter list. `https://easylist.to/easylist/easylist.txt` [Downloaded: March 25th, 2024].

[29] EasyList. 2024. EasyPrivacy - tracking filter list. `https://easylist.to/easylist/easyprivacy.txt` [Downloaded: March 25th, 2024].

[30] S. J. Eravuchira, V. Bajpai, J. Schönwälder, and S. Crawford. 2016. Measuring Web Similarity from Dual-stacked Hosts. In *Proc. International Conference on Network and Service Management (CNSM).*

[31] e. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner. 2014. Web Data Extraction, Applications and Techniques: A Survey. *Knowledge-Based Systems* 70 (November 2014), 301–323.

[32] R. Fielding, M. Nottingham, and J. Reschke. 2022. *HTTP Semantics.* RFC 9110. Internet Engineering Task Force.

[33] R. Fielding, M. Nottingham, and J. Reschke. 2022. *Hypertext Transfer Protocol (HTTP/1.1).* RFC 9112. Internet Engineering Task Force.

[34] C. Filsfils, S. Previdi, L. Grinsberg, B. Decraene, S. Likowski, and R. Shakir. 2018. *Segment Routing Architecture.* RFC 8402. Internet Engineering Task Force.

[35] G. Fioccola, P. Volpato, J. Palet Martinez, G. Mishra, and C. Xie. 2023. *IPv6 Deployment Status.* RFC 9386. Internet Engineering Task Force.

[36] K. Fujiwara, A. Sato, and K. Yoshida. 2013. DNS Traffic Analysis – CDN and the World IPv6 Launch. *Journal of Information Processing* 21, 3 (July 2013), 517–526.

[37] F. Golkar, T. Dreibholz, and A. Kvalbein. 2014. Measuring and Comparing Internet Path Stability in IPv4 and IPv6. In *Proc. International Conference and Workshop on the Network of the Future (NOF).*

[38] Google. 2024. IPv6 deployment statistics. (2024). `https://www.google.com/intl/en/ipv6/statistics.html` [Last Accessed: May 27th, 2024].

[39] HTTPArchive. 2012-2024. Report: State of the Web, Total Requests. https://httparchive.org/reports/state-of-the-web#reqTotal [Last Accessed: May 8th, 2024].

[40] C. Huitema. 2006. *Teredo: Tunneling IPv6 over UDP through NATs*. RFC 4380. Internet Engineering Task Force.

[41] G. Huston. 2013–2024. *IPv4 Address Report*. https://ipv4.potaroo.net [Last Accessed: May, 8th 2024].

[42] G. Huston. 2015. Examining IPv6 Performance. https://labs.ripe.net/author/gih/examining-ipv6-performance/ [Last Accessed: February 17th, 2024].

[43] G. Huston. 2015. Is IPv6 Faster. https://ripe71.ripe.net/archives/video/1219/ [Last Accessed: March 17th, 2024].

[44] G. Huston. 2016. Evaluating IPv4 and IPv6 Packet Fragmentation. https://labs.ripe.net/author/gih/evaluating-ipv4-and-ipv6-packet-fragmentation/ [Last Accessed: May 13th, 2024].

[45] S. Ihm and V. S. Pai. 2011. Towards Understanding Modern Web Traffic. In *Proc. ACM Internet Measurement Conference (IMC)*.

[46] J. Iyengar and M. Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. Internet Engineering Task Force.

[47] Juniper. 2023. *Understanding Dual Stacking of IPv4 and IPv6 Unicast Addresses*. https://www.juniper.net/documentation/us/en/software/junos/is-is/topics/concept/ipv6-dual-stack-understanding.html [Last Accessed: May, 8th 2024].

[48] G. Kacmarcik and T. Leithead. 2024. *UI events*. W3C Working Draft. W3C: The World Wide Web Consortium. https://www.w3.org/TR/uievents/#event-type-load [Last Accessed: March 19th, 2024].

[49] M. Kruisselbrink. 2023. *File API*. W3C Working Draft. W3C: The World Wide Web Consortium. https://www.w3.org/TR/FileAPI/ [Last Accessed: March 17th, 2024].

[50] M. Larson and Contributors. 2024. urllib3 python library. https://pypi.org/project/urllib3/ [Last Accessed: May 24th, 2024].

[51] L. Masinter. 1998. *The data URL scheme*. RFC 2397. Internet Engineering Task Force.

[52] MaxMind. 2024. *Localize content with GeoIP.* https://www.maxmind.com/ [Last Accessed: May 4th, 2024].

[53] M. Nikkhah and R. Guérin. 2015. Migrating the Internet to IPv6: an exploration of the when and why. *IEEE/ACM Transactions on Networking (ToN)* 24, 4 (August 2015), 2291 – 2304.

[54] OVHcloud France. 2024. : Cloud Computing and Web Hosting. https://www.ovhcloud.com/ [Last Accessed: April 28th, 2024].

[55] M. Piraux and O. Bonaventure. 2023. *Adaptive Address Family Selection for Latency-Sensitive Applications on Dual-stack Hosts.* cs.NI 2309.05369. arXiv.

[56] Python Software Foundation 2024. *Python 3.12.3 Docs: socket — Low-level networking interface.* Python Software Foundation. https://docs.python.org/3/library/socket.html [Last Accessed: May 24th, 2024].

[57] Python Software Foundation 2024. *Python 3.12.3 Docs: sys — System-specific parameters and functions.* Python Software Foundation. https://docs.python.org/3/library/sys.html [Last Accessed: May 24th, 2024].

[58] D. Schinazi and T. Pauly. 2017. *Happy Eyeballs Version 2: Better Connectivity Using Concurrency.* RFC 8305. Internet Engineering Task Force.

[59] Selenium. 2023. Selenium automates browsers. That's it! https://www.selenium.dev [Last Accessed: February 19th, 2024].

[60] J. Sengupta, T. Shreedhar, D. Nguyen, R. Kramer, and V. Bajpai. 2024. *Through the Lens of Google CrUX: Dissecting Web Browsing Experience Across Devices and Countries.* cs.NI 2308.06409. arXiv.

[61] S. Stewart and D. Burns. 2024. *WebDriver Protocol Standard.* W3C Working Draft. W3C: The World Wide Web Consortium. https://www.w3.org/TR/webdriver2/ [Last Accessed: March 17th, 2024].

[62] D. Thaler, R. P. Draves, A. Matsumoto, and T. Chown. 2012. *Default Address Selection for Internet Protocol Version 6 (IPv6).* RFC 6724. Internet Engineering Task Force.

[63] G. van der Meer. 2024. Chrome Web Store: I Still Don't Care About Cookies. https://chrome.google.com/webstore/detail/i-still-dont-care-about-c/edibdbjcniadpccecjdfdjjppcpchdlm [Last Accessed: May 24th, 2024].

[64] A. van Kesteren. 2024. *HTML Standard.* Living Standard. WHATWG: Web Hypertext Application Technology Working Group. https://html.spec.whatwg.org/multipage/ [Last Accessed : March 17th, 2024].

[65] A. van Kesteren. 2024. *URL Standard*. Living Standard. WHATWG: Web Hypertext Application Technology Working Group. `https://url.spec.whatwg.org` [Last Accessed: March 17th, 2024].

[66] VULTR. 2024. The Everywhere Cloud. `https://www.vultr.com/` [Last Accessed: April 28th, 2024].

[67] D. Wing and A. Yourtchenko. 2012. *Happy Eyeballs: Success with Dual-Stack Hosts*. RFC 6555. Internet Engineering Task Force.

[68] Yutian Y., Yunhui Z., and Xinyue Liu. 2023. ADHERE: Automated Detection and Repair of Intrusive Ads. *Proc. IEEE/ACM International Conference on Software Engineering (ICSE)* 24, 4 (May 2023), 486 – 498.

Appendix A: NetQuartz implementation details

As explained in Section 3.6, several patches have to be performed during NetQuartz development.

The first (see Figure A) is a patch on the *socket.getaddrinfo()*, it gives us the opportunity to select to select either full-`IPv4` or `IPv6`-preferenced behavior:

The *urllib3* library does not natively expose the `IP` address used in the `HTTP` connection. To capture this information, we modified the *_make_request* method within the *HTTPConnectionPool* class such that each response object now includes the **peer** attribute, which stores the `IP` address and port number of the server to which the connection was established. The `Python` code for this patch is represented through Figure B.

The timing management problem of the `Python` *requests* library has been handled by using a trace function, as explained in Section 3.6.2. The code used to implement this trace function is highlighted through Figure C.

```python
old_getaddrinfo = socket.getaddrinfo

def new_getaddrinfo(*args, **kwargs):
    responses = old_getaddrinfo(*args, **kwargs)
    return [response
            for response in responses
            if response[0] == socket.AF_INET]

# Calling `socket.getaddrinfo = new_getaddrinfo` will select full-IPv4 behavior
# Calling `socket.getaddrinfo = old_getaddrinfo` will get back to IPv6-preferenced behavior
```

Figure A: Python patch performed on *getaddrinfo()* to select IP address family preference.

```python
from urllib3.connectionpool import HTTPConnectionPool
def _make_request(self,conn,method,url,**kwargs):
    response = self._old_make_request(conn,method,url,**kwargs)
    sock = getattr(conn,'sock',False)
    if sock:
        setattr(response,'peer',sock.getpeername())
    else:
        setattr(response,'peer',None)
    return response
HTTPConnectionPool._old_make_request = HTTPConnectionPool._make_request
HTTPConnectionPool._make_request = _make_request
# source: https://stackoverflow.com/questions/22492484/how-do-i-get-the-ip-address-from-a-http-request-using-the-requests-library
```

Figure B: Python patch performed on *urllib3* to fetch the IP address used while establishing the connection.

```python
def timeout_trace(frame, event, arg):
    if time.time() - start_time_timeout > resource_timeout:
        raise TimeoutError("Operation timed out")
    return timeout_trace
# resource_timeout being set to 2 seconds, start_time_timeout is passed to the trace function when the HTTP GET request is executed, and simply corresponds to the timestamp of when the request was initiated.
```

Figure C: Python trace function to measure total resource downloading time against resource_timeout.

Appendix B: Paper submitted at the Internet Measurement Conference (IMC) 2024

The study of this work led us to submit a paper to the Internet Measurement Conference (IMC) 2024. The paper is currently under review (early reject notification on June 28th, 2024; notification of acceptance on July, 31st 2024). The paper is essentially a compressed version of this thesis, with fewer technical details and fewer results.

NetQuartz: Dissecting Dual-Stack Website Content

Paper #181, 12 pages body, 14 pages total

ABSTRACT

The ongoing shift from IPv4 to IPv6 is crucial for the evolving Internet landscape. However, the prevalence of dual-stacked environments requires a deeper understanding of how web content is distributed across both protocols, especially given the varied nature of websites configurations. While prior studies have focused on IPv6 adoption and performance metrics, limited research explores dual-stacked servers' impact on the application layer where user experience is directly affected.

To address this gap, we developed NetQuartz, a tool designed to asses the performance and the content delivery of websites across dual-stack servers. We have deployed NetQuartz over several vantage points and collected data for more than 200,000 websites, revealing patterns by classifying them into three classes based on server configurations: (*i*) Fully IPv4 websites, where all resources are loaded from IPv4-only servers; (*ii*) Fully Dual-Stacked websites, which load all their resources from servers supporting both IPv4 and IPv6; (*iii*) Mixed websites, which retrieve resources from a combination of both server types. *Script* and *Image* were identified as the two most dominant categories in terms of content, regardless of the website's class. Fully IPv4 websites generally contained fewer resources, leading to smaller page sizes and faster load times. Fully Dual-Stacked and Mixed configuration websites showed better performance under IPv6-preferenced loading, despite Mixed websites tending to load more resources over IPv4. NetQuartz and collected data will be made available to the research community upon paper acceptance.

KEYWORDS

dual-stack, IPv6, performance, content, NetQuartz

1 INTRODUCTION

Since the early 80's, the Internet Protocol version 4 (IPv4) [13] has been the main protocol supporting packet switching networks and internetworking technology. However, the exponential deployment of the Internet, since the mid-90's, has led to an exhaustion of IPv4 addresses [30], requiring a transition towards its successor, Internet Protocol version 6 (IPv6) [15]. Since then, IPv6 has been more and more adopted [11, 12, 17, 28]. If IPv6 allows for dealing with IPv4 address exhaustion [30], it also comes with an additional feature, called *Extension Header* [10, 16], that leads to more flexibility and innovation. Examples of innovations can be found in observability [7] or forwarding [26].

A smooth transition from IPv4 to IPv6 has been made possible thanks to *dual-stack* devices, i.e., devices with network interfaces that can originate and understand both IPv4 and IPv6 packets [35], avoiding so a complete shift from one protocol version to the other. Up to now, studies primarly focus on the adoption and basic performance of IPv6 metrics, such as addressing and routing [11, 12, 17, 39]. However, few researches focused on dual-stack aspects. For instance, Bajpai et al. [3, 4, 22] provide valuable insights into server performance and transport-layer metrics but do not extend their analysis to the application layer where user experience directly takes place. This is crucial as a significant portion of websites includes content fetched from multiple, distinct servers [8], possibly over different version of the Internet Protocol, introducing so additional delay. Huston [31, 32] used Google ads to measure the latency of client connections to APNIC servers and found that IPv6 was faster than IPv4 half the time, but experimented with a higher failure rate. Further, Bajpai et al. [28], show a significant reduction in latency for both IPv4 and IPv6 for Alexa top 10K websites. They also report a reduction in IPv6 connection failures and find that the main contributors to these failures are *Image*, *Stylesheet*, *Script* categories. Also, Dhamdhere et al. [17] evaluated these load times on the Alexa list and found that IPv6 performance can be much worse than IPv4 if the AS paths are different, otherwise they are similar. However, it is unclear which amount of content (e.g., *Image*, *Stylesheet*, *Script*, etc.) is delivered through wich version of the Internet Protocol.

This is exactly what we want to investigate in this paper. In particular, we make the following contributions: first, we introduce NetQuartz, our tool designed to assess and compare the content delivery capabilities and performance of websites across dual-stack servers. For each website, NetQuartz maintains two loading sessions: (*i*) a full-IPv4 loading, always preferring IPv4 connections, and (*ii*) an IPv6-preferenced loading, prioritizing IPv6 connections where available, and defaulting to IPv4 otherwise. This paper carefully explains how NetQuartz has been built to reach those goals.

Second, we have deployed NetQuartz over several vantage points and collected data from more than 200,000 analyzable websites extracted and filtered from DomCop list [20].

Third, we propose to classify websites based on their server configurations into three classes: (*i*) Fully IPv4 websites, where all resources are loaded from IPv4-only servers; (*ii*) Fully Dual-Stacked websites, which load all their resources from servers supporting both IPv4 and IPv6; (*iii*)

Mixed websites, which retrieve resources from a combination of both server types. All three classes are initially accessed with full-IPv4 loading, using only IPv4 connections. Separately, Mixed and Fully Dual-Stacked websites are also accessed with an IPv6-preferenced loading, which prioritizes IPv6 connections where possible. For Fully Dual-Stacked websites, IPv6-preferenced is equivalent to full-IPv6 loading, as all resources are available on dual-stack servers.

Fourth, whatever the website class, content-related analysis revealed that *Image* and *Script* were the predominant categories, both in terms of the size and the number of resources queried over HTTP/S. The analysis also showed that Fully IPv4 websites typically load fewer resources, resulting in faster loading times. For Fully Dual-Stacked and Mixed websites, the performance under IPv6-preferenced loading consistently outperforms that of IPv4. We observed that Mixed websites load 20% faster in median and 11% faster in their 95th percentile when using IPv6-preferenced loading, Fully Dual-Stacked websites load 16% faster in median and 24% faster in the 95th percentile. The observation is particularly interesting with Mixed websites, for which we found that 64% of these websites were loading more HTTP/S resources over IPv4 than IPv6, yet the overall performance in IPv6-preferenced sessions remains more efficient.

Finally, NetQuartz source code, as well as data collected, will be made freely available to the research community upon paper acceptance.

The remainder of this paper is organized as follows: Sec. 2 provides a detailed explanation of NetQuartz and its methodology; Sec. 3 discusses the data collection methodology, explaining the setup and execution of our measurement environment; Sec. 4 presents our results, first exploring the delivery of web content across websites, then examining the specific dual-stack distribution of Mixed websites, and finally assessing broader performance measures. Sec. 5 positions our work with respect to the state of the art; finally, Sec. 6 concludes this paper by summarising its main achievements and suggesting areas for future researches.

2 NETQUARTZ

NetQuartz, our tool for measuring content delivery and performance of websites, works in three steps as illustrated in Fig. 1:

Step 1 : Main HTML domain processing (Sec. 2.1). It performs a DNS lookup for the domain and send ping towards the server hosting the main HTML of the website;

Step 2 : Resources collection (Sec. 2.2). It collects (through an headless browser technology) and logs all the resources and their categories (*Image*, *Script*, etc) that are loaded along with the main HTML document;

Step 3 : Website loading comparison protocol (Sec. 2.3). It loads the full website both in an full-IPv4 and an IPv6-preferenced loading and collects metrics from the loading of each resource.

2.1 Step 1: Main HTML Domain Processing

As depicted in Fig. 1a, NetQuartz begins by accepting a list of domains along with the URLs of a web home page that can be queried on that domain and the HTTP status code returned while fetching the home page. The domain of these URLs point to the server that hosts the main HTML document of the website. Only entries with successful status code (200) are considered for the processing.

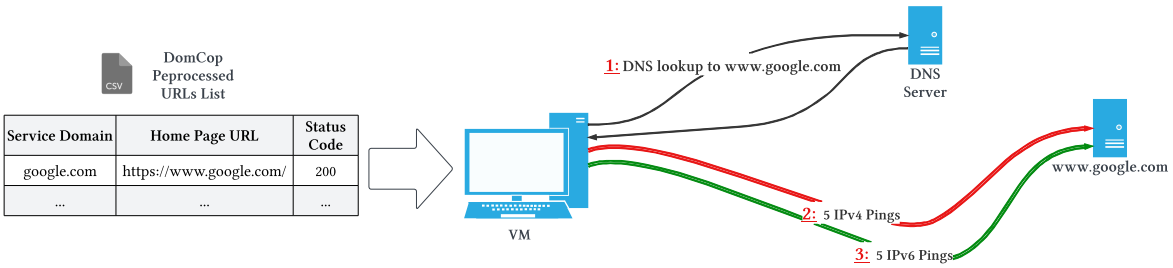
NetQuartz starts by querying the DNS to assess the nature of the main HTML hosting server and then performs ping in IPv4 (and IPv6 if the server has been identified as dual-stack) on the first record returned by the DNS. NetQuartz logs the average time and standard deviation associated with these pings, sending a predefined number of ICMP packets to each home page's hosting server. The goal is to have an estimation of the RTT (in both IPv4 and IPv6 when available) towards this main server, it will thus be possible to determine at a later stage whether this metric is a sufficient indicator of website performance.

2.2 Step 2: Resources Collection

Most websites not only load the main HTML for the home page but will also parse it [47] to load additional resources (*Image*, *Script*, etc) so that they can ensure the website complete rendering. NetQuartz relies on Selenium [43], a headless browser technology, for facilitating the collection of these resources. Fig. 1b illustrates this step: we use a Chrome browser through a dedicated executable driver [19], allowing script-driven browser interactions. The communication between Selenium and the chromedriver is handled through the WebDriver protocol, an HTTP-based protocol that uses JSON objects to instruct the browser what actions to perform [45]. These resources are kept on memory for further analysis.

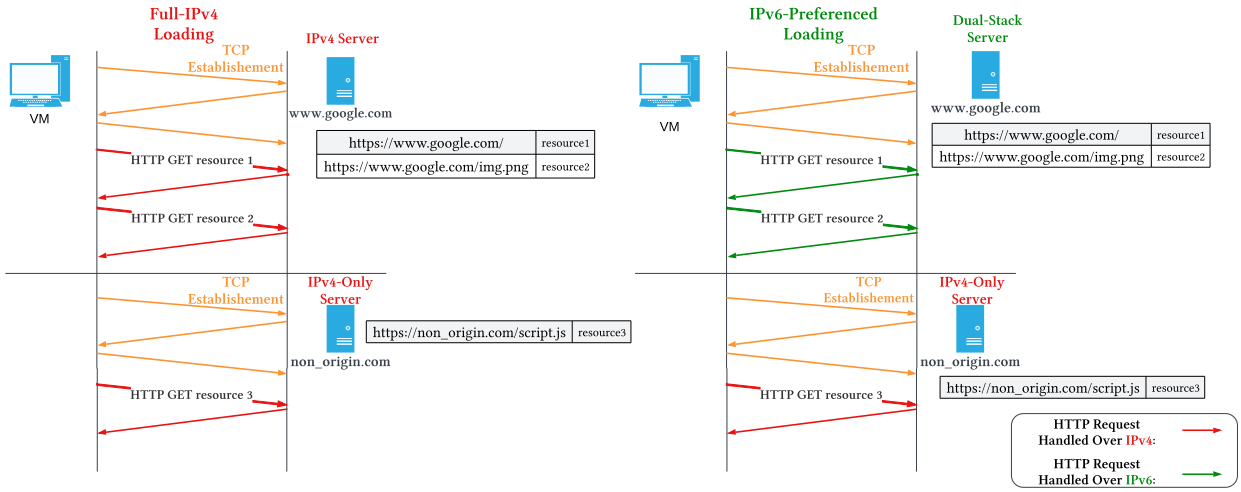
2.3 Step 3: Website Loading Comparison Protocol

Before establishing the methodology for comparing websites, it is worth looking at the generic behaviour of a dual-stack user when loading a website from a set of servers. The browser first performs a DNS lookup on the domain hosting the main HTML document, the latter is then retrieved via an HTTP/S request. As soon as the main HTML is available, the browser parses it to identify all the resources needed to render the web page, which may come from servers different than the one hosting the main HTML. When resources are



(a) Step 1: main `HTML` domain processing.

(b) Step 2: resources collection.



(c) Step 3: website loading comparison protocol.

Figure 1: NetQuartz general overview.

available in both `IPv4` and `IPv6`, the default behaviour of most browsers is often to prefer `IPv6` [46]. This can be problematic if the `IPv6` connectivity is broken. It is for this reason that most browsers now implement the Happy Eyeballs algorithm, which recommends selecting one address family to establish the connection, as a result of a competition started after a specific timer (typical value is 300 ms for Chrome and Firefox) [42, 50]. For practical reasons and in order to focus only on the comparison between `IPv4` and `IPv6` and not on an efficient selection of the protocol, the Happy Eyeballs algorithm is not included in our protocol.

Fig. 1c catalogs our protocol for comparing website loading times, highlighting the complexity of web resource downloads, which often originate from servers distinct from the origin domain. The protocol takes as input the list of resources collected with `Selenium` during step 2 (Sec. 2.2) and mimics as much as possible a user browsing behavior, adhering to the `HTTP` standard [24, 25] and connection persistence.

Sequential requests through session management showcase sessions with full-IPv4 loading or an IPv6-preferenced loading, illustrating that prioritizing IPv6 does not guarantee exclusive IPv6 resource loading, as depicted in the figure.

Practically, this preference is enforced by manipulating the getaddrinfo() function to return only IPv4 addresses during a full-IPv4 loading while retaining default behavior for an IPv6-preferenced one. To mitigate biases between the two loading strategies, a DNS query precedes each HTTP/S GET, ensuring domain name server caching before the loading times measurements.

During each HTTP/S GET request, additional metrics are collected, such as the resource URL and the *content-type* HTTP header. For both full-IPv4 and IPv6-preferenced loadings, we measure the load time of each resource, we also collect the byte size of the HTTP/S response, the status code of the response, and information about any resource download errors. All the data is logged into a json file and kept for further postprocessing. The total website loading time is computed by summing the loading times of all successfully loaded resources within both IPv4 and IPv6-preferenced loadings, ensuring neither loading strategy is unfairly penalized for experiencing more failures or timeouts. To preserve data integrity at this level, failure reasons are logged for later comparison, aiming to analyze the failure rate (and reasons) of web resources within full-IPv4 loading versus IPv6-preferenced loading.

NetQuartz proactively identifies and handles specialized URLs [48], such as data URL that encodes data directly within the URL itself [37], these resources do not involve traditional network loading over IPv4 or IPv6. They are still logged for postprocessing (Sec. 3.4), including the assessment of the size and category of data encapsulated within data URL resources.

2.4 Hyperparameters

The first hyperparameter – related to main HTML domain processing (Sec. 2.1) – involves the count of IPv4 (and IPv6 if applicable) pings performed without compromising the tool's performance. The value chosen for this parameter will be discussed later in Sec. 3.3.

The majority of websites dynamically load a significant portion of their content via the *Fetch* API and/or the XMLHttpRequest (*XHR*) API [34]. This introduces a degree of ambiguity to the concept of a website's loading process, theoretically allowing it to extend indefinitely. To detect full initial loading, Selenium uses the browser's "*load*" event– a pivotal moment in a webpage's lifecycle signaling the full loading and parsing of the page [36]. This will not take into account content dynamically loaded post-initial "*load*" event via asynchronous calls.

That brings us into our second hyperparameter: we have implemented a timer "selenium_timeout" for the web loading process conducted by Selenium (Sec. 2.2). Similarly, another timer "resource_timeout" is used within the third NetQuartz step (Sec. 2.3), applying to the loading time of each resource so that we prevent resources from loading indefinitely or in case the targeted web server is not even responding. Upon expiration, this timer exposes the timeout's cause and context: a "*connect timeout*" denotes a failed server connection attempt, whereas a "*read timeout*" occurs upon a failed read attempt from an already established connection.

Another critical parameter is the data write rate to disk, introducing a protective layer for the tool by logging data at a specific frequency. The last parameter relates to a more global timeout "processing_timeout" that limits the total NetQuartz processing time for each domain. This is intended to prevent excessive time spent on sites with an abundance of resources that exceed a predefined threshold. If the total time exceeds the set limit, domain processing stops, but data collected is kept, ensuring usability while acknowledging the incomplete analysis of all site resources. The values chosen for these parameters will be discussed later in Sec. 3.3.

3 DATA COLLECTION METHODOLOGY

Fig. 2 provides an overview of the methodology followed when running NetQuartz for collecting data. As illustrated, a four steps methodology is applied, each of those steps being described in the following subsections.

3.1 Step 1: Domains List Collection

Due to the discontinuation of the Alexa list, we rely on DomCop [20] which lists domain services (e.g., google.com). The list contains 10 million entries but we limit ourselves to the the first 300,000 websites for the processing and the analysis (list obtained on February 19th, 2024). As explained in Sec. 2.1, NetQuartz needs as input not only the main service domain, but also the web home page URL for that service, along with the HTTP status code fetched. These pieces of information are not included by default in the DomCop file, requiring so a preprocessing stage (Sec. 3.2).

3.2 Step 2: Preprocessing Step

An initial HTTPS connection attempt to each website is made, following HTTP redirection codes (range 300 to 399) until a successful 200 status code is obtained. This redirection strategy is commonly employed by web services to guide users directly to the service's web home page (e.g., navigating to google.com in a browser redirects to https://www.google.com/). If the HTTPS attempt fail, a subsequent attempt over HTTP is made. This process allows us to determine the applicable

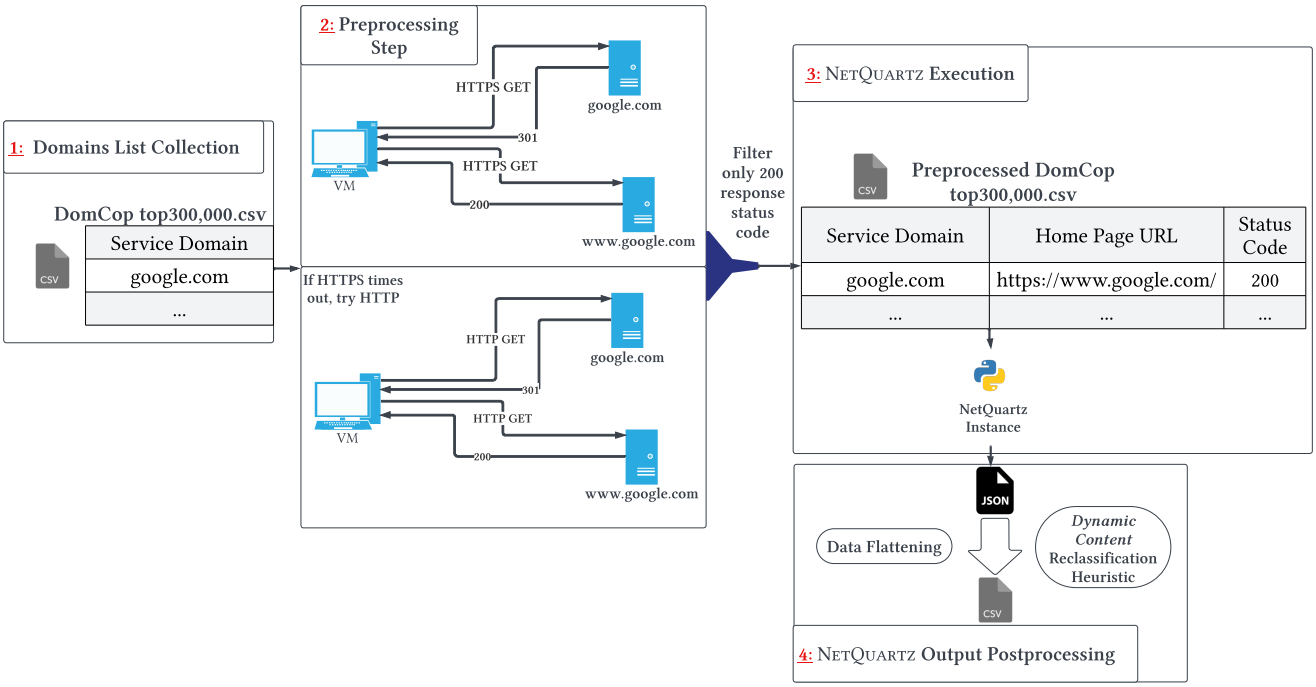


Figure 2: Data collection methodology overview.

Category	Total		HTTP		HTTPS	
	Raw	Prop.	Raw	Prop.	Raw	Prop.
All Websites	300,000	1.0	15,276	0.051	266,019	0.887
200 (Success) Status Code	217,974	0.727	9,264	0.031	208,710	0.696
Status Code ≠ 200	63,321	0.211	6,012	0.02	57,309	0.191
No Response	18,705	0.062	/	/	/	/

Table 1: Summary of the preprocessing step (total: 300,000 websites).

application layer protocol, the URL for accessing the service's home page (e.g., `https://www.google.com/`), and its status; all such information is appended to the original `csv` file.

The preprocessing phase results are summarized in Table 1: 73% of input websites yielded a home page with a 200 (OK, request succeeded) status code, 21% responded with error status codes, while 6% did not respond to connection requests. Despite a preference for HTTPS connections, 4% of the sites are still exclusively accepting HTTP connections.

3.3 Step 3: NetQuartz Execution

We deployed NetQuartz on three vantage points (VP) that are hosted by different cloud service providers across three regions, one in North America hosted by *Digital Ocean* [18], one in Europe hosted by *OVH* [40], and one in Asia hosted by *Vultr* [49]. Each VP was tasked with allocated portions of the `csv` resulting from previous steps: North America and Asia (both got 1 CPU and 0.8 GB of RAM) dealed each one with 25,000 randomly selected items from the `csv`, while the European VP (8 CPUs and 32 GB of RAM) run through 250,000 websites randomly picked up from the `csv`. The measurement campaign was launched on March 2nd, 2024 and concluded on March 25th, 2024.

The values fixed for each NetQuartz's hyperparameter (Sec. 2.4) are as follows: `selenium_timeout` has been fixed to 40 seconds, `resource_timeout` to 2 seconds, `processing_timeout` to 70 seconds. During NetQuartz execution, data logging to disk was performed by each VP after every 1,000 websites processed. Five `IPv4 pings`– and five `IPv6 pings` when applicable – were executed towards the main HTML hosting server (Fig. 1a). Those values were fixed based on the observation of Cumulative Distribution Functions detailed in Appendix A.

NetQuartz execution summary is reported in Table 2 which indicates that of the 217,974 websites accepted by the preprocessing stage, 93.2% encountered no errors, 1.5% triggered the `selenium_timeout`, and 5.3% had their processing manually halted upon exceeding the `processing_timeout` though these sites remain analyzable (see Sec. 2.4), with an equitable assessment of their resources between full-IPv4 and IPv6-preferenced sessions.

Processing Completed		selenium_timeout		processing_timeout	
Raw	Prop.	Raw	Prop.	Raw	Prop.
203,102	0.932	3,329	0.015	11,543	0.053

Table 2: Summary of the NetQuartz execution step (total: 217,974 websites).

3.4 Step 4: NetQuartz Output Postprocessing

When NetQuartz processing is done on each VP, `json` outputs are merged, parsed, and flattened into a final `csv` file. During this postprocessing stage, we apply a content reclassification heuristic to fill a gap in the classification of the `Chrome` performance tool (Sec. 3.4.1). This stage is also responsible for processing `data URL` resources logged by NetQuartz (Sec. 2.3) to infer their types and sizes.

3.4.1 Reclassifying Resources Loaded Through Fetch & XHR API. While straightforward `HTML` document parsing techniques might suffice when classifying static content, they fall short in capturing the full spectrum of dynamic web content [23]. `Selenium` makes us able to operate the network performance tool, accessible through the `Chrome` driver interface, to infer the following categories: *Document*, *Image*, *Media*, *Fetch*, *XHR*, *Font*, *Stylesheet*, *Script*, and *Other*. That last category is a kind of tie-break, for resources that do not fall within one well known category.

`Chrome`'s network performance tool does not explicitly reveal resource categories for *Fetch & XHR*, they represent more a classification of requests within the dynamic context of a site. This categorization gap comes from the `Chrome` performance tool's dual consideration of content type and loading context, with dynamic loading contexts (*Fetch & XHR*) taking precedence over content-based categories.

Addressing this, we use the *content-type* header collected by NetQuartz (Sec. 2.3) to build an heuristic able to reclassify the *Fetch & XHR* context-related categories into the content-related ones. The *content-type* `HTTP` Header reliability is not without its challenges; services might declare a content type that diverges from the actual content, a practice countered by browsers through `MIME` (Multipurpose Internet Mail Extensions) type sniffing [21]. This involves payload inspection – following well-defined standards [1] – to derive the true content type, potentially influenced by the resource's file extension. Ethically, our developed heuristics refrain from inspecting payload content.

The basis of the heuristic is keyword detection within the *content-type* header encoding the resource `MIME` type, described by type, subtype, and parameters:

```
type/subtype;parameter=value
```

Keyword detection focuses on the type and subtype fields, and is summarized in Fig. 3. Validation and evaluation of the heuristic has been performed and documented in Appendix B.1.

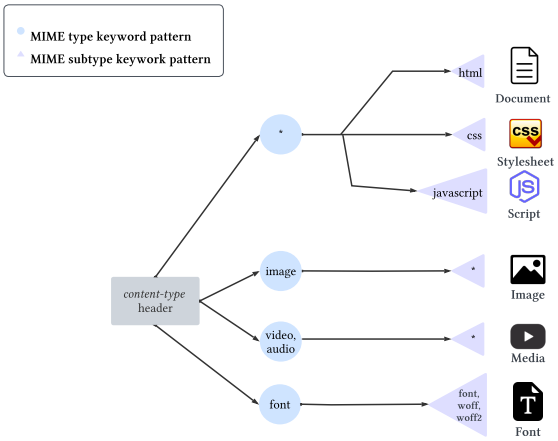


Figure 3: Resource classification heuristic, based on the *content-type* header of the `HTTP` response. Asterisks ('*') indicate no particular emphasis on the field.

4 RESULTS

This section presents results of data collected by NetQuartz. Prior to results discussion, we propose a classification of websites according the way they respond to NetQuartz requests:

- Class 1: Websites loading all their resources over `IPv4`-only servers can only be accessed and loaded via the `IPv4` protocol. We choose to label such websites as `Fully IPv4` websites.
- Class 2: Websites loading all their resources over dual-stack servers are fully accessible over `IPv6`, but full access over `IPv4` is also possible. We label such websites as `Fully Dual-Stacked` websites.
- Class 3: We have observed websites hosting some resources on `IPv4`-only servers and others on dual-stack ones, supporting both `IPv4` and `IPv6`. This setup allows those website to be accessed either entirely over `IPv4` or through a combination of both protocols. We label such websites as `Mixed` websites.

All website classes can be loaded using a full-`IPv4` approach (see Fig. 1c). However, for `Fully Dual-Stacked` websites, `IPv6`-preferenced loading corresponds to be a full-`IPv6` approach, as all the resources are hosted on dual-stack servers. Finally, `Mixed` websites can be accessed using an `IPv6`-preferenced loading approach, by preferring `IPv6` connections where available. Fig. 4 provides a summary of the different website classes along with the content loading approaches.

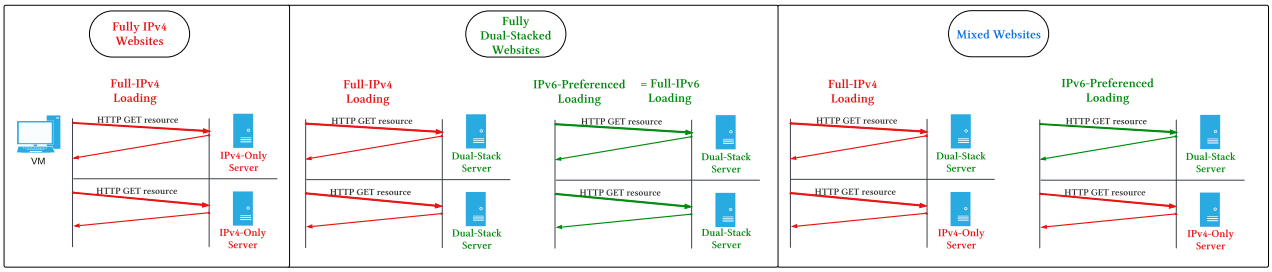
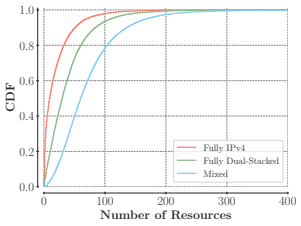


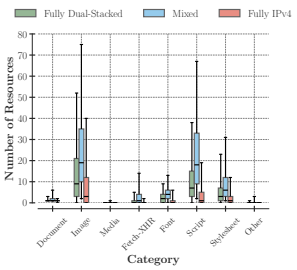
Figure 4: Website classification summary.

Mixed		Fully IPv4		Fully Dual-Stacked		No Resources	
Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.
162,190	0.7444	24,354	0.112	27,936	0.128	3,494	0.016

Table 3: Distribution of the different website configurations (total: 217,974 websites).



(a) Total number of resources requested.



(b) Number of resources requested by category.

Figure 5: Number of resources requested via an HTTP/S GET during the loading of Fully IPv4, Fully Dual-Stacked, and Mixed websites. The bottom and top whiskers represent the 5th and 95th percentiles, respectively. For readability reasons, outliers are not shown in the boxplots.

Table 3 provides an initial overview of the distribution of websites based on their class. More than 74.4% of the measured websites belong to the Mixed class. Both Fully IPv4 and Fully Dual-Stacked classes are equally represented (11.2% and 12.8% respectively). Finally, Table 3 shows a low proportion (1.6%) of websites from which no data could be captured using Selenium.

In the following, we first dissect data collected by looking at the web pages content (Sec. 4.1). Second, Sec. 4.2 focuses on the most prominent class, i.e., Mixed. Finally, Sec. 4.3 looks at the three classes performance.

4.1 Web Content Delivery

We look at the content delivery. In particular, we dissect the web pages collected by NetQuartz such that we expose the various *resources* composing home pages.

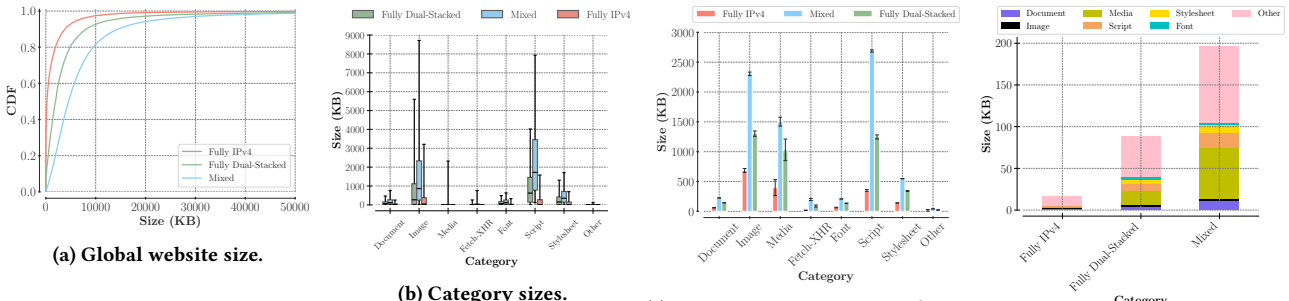
Our investigation into resource distribution directly corresponds to the number of HTTP/S GET requests initiated by a user. This metric is obviously independent of the IP protocol used for loading.

Fig. 5a shows, as a CDF, the number of resources contained in web pages, in function of the website class. The Fully IPv4 class consistently load fewer resources, typically loading up to 70 resources in their 95th percentile. The Mixed class requires more resources, up to 168 in the 95th percentile. Finally, the Fully Dual-Stacked class is between the two other classes. This difference can be explained by the fact that websites tend to become larger in size and number of resources fetched over time. Considering HTTP Archive [29] data collected between 2012 and 2024, we find that the number of requests performed towards websites has increased by 25% in the median since 2012. As stated in Sec. 1, the IPv6 adoption has also significantly increased over time and it is thus interesting to note that Fully IPv4, being older than Fully Dual-Stacked and Mixed websites, will therefore load fewer resources.

Fig. 5b splits the downloaded resources into their respective categories, according to the website class. The distribution shows that *Image* and *Script* are the predominant categories, regardless of the website class. *Media* and *Other* categories are barely nonexistent.

Our analysis extends into the overall size of websites, noting that the total raw – meaning after decoding – data transmitted during the loading process is totally independent of whether the loading is IPv4-only or IPv6-preferenced.

As illustrated in Fig. 6a, Fully Dual-Stacked websites have the highest data size, which correlates with their higher resource numbers. Fig. 6b shows that *Image* and *Script* are still the predominant categories, the figure also reveals a significant insight about resource utilization and efficiency,



(a) Global website size.

(b) Category sizes.

(c) Average category size with 95% confidence intervals measures.

(d) *Fetch & XHR* reclassification.

Figure 6: `Fully IPv4`, `Fully Dual-Stacked`, and `Mixed` websites raw size distribution (in Kilobytes). The bottom and top whiskers represent the 5th and 95th percentiles, respectively. For readability reasons, outliers are not shown in the boxplots.

particularly in `Mixed` websites, where *Script* resources, although fewer in number, contribute disproportionately to the total data volume. This can be attributed to the complexity and functionality embedded within those scripts, which often include libraries and frameworks that are essential for modern web applications but are inherently large.

In terms of data distribution, average values provide an interesting perspective. While the majority of websites manage to maintain *Media* sizes relatively low, as indicated by the clustering of data points near the origin in the boxplot (Fig. 6b), there are outliers with considerably large resource loads. These outliers skew the average upwards, as depicted in Fig. 6c. This observation underscores the variable nature of web content distribution, where most websites optimize resource sizes, but some, possibly due to specific functional or design requirements, load larger *Media* files.

Fig. 6d summarizes the reclassification step (Sec. 3.4.1) of the resources loaded through the *Fetch & XHR* API. This is done by extracting the average size of the *Fetch & XHR* category and subsequently illustrating how each other category contributes to this average. On average, *Media* emerges as the largest category typically loaded asynchronously. This asynchronous loading allows for enhanced user experiences by not blocking the rendering of other page elements while large media files are being loaded.

4.2 Dual-Stack Content Distribution

The `Mixed` websites are those which, when loaded with the default behavior of a dual-stack client (i.e. to prefer `IPv6`), load a certain portion of their resources in `IPv6` and the other in `IPv4`. The aim of this section is to focus on the 162,190 so-called `Mixed` websites (see Table 3) and to evaluate the portion of content downloaded in `IPv6` and `IPv4` when `IPv6`-preferenced loading occurs.

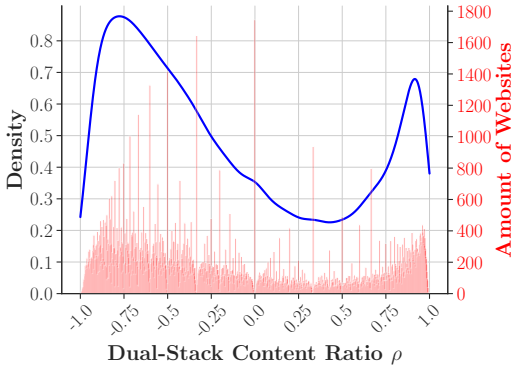


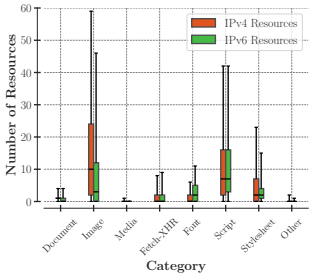
Figure 7: Normalized ratio (ρ) calculated for the 162,190 `Mixed` wesbites, prioritizing more resources loaded over `IPv4` on the left-hand side (negatives values), and more resources loaded over `IPv6` on the right-hand side (positive values).

Fig. 7 presents the probability density function of a normalized ratio calculated for each website. We express the ratio as

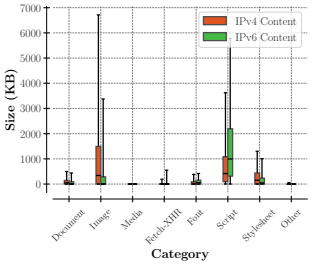
$$\rho = \frac{\texttt{IPv6\_}resources - \texttt{IPv4\_}resources}{total\_number}. \quad (1)$$

where $\rho \in]-1, 1[$. The limits of the interval (-1 and 1) are rejected as they represent the case of `Fully IPv4` and `Fully Dual-Stacked` websites respectively. A negative ρ means the website prefer to load more resources over `IPv4`, while a positive ρ means the website prefer `IPv6`. A null value means that the website equally loads resources in `IPv4` and `IPv6`.

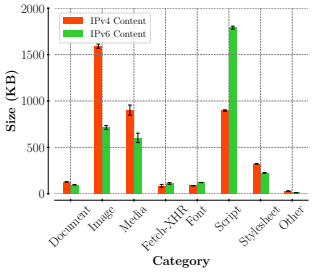
Fig. 7 shows that 64% of the data points fall into negative values, indicating a higher number of resources loaded over



(a) Distribution of resources loaded over IPv4 and IPv6 for the different categories.



(b) IPv4 content size and IPv6 content size (in Kilobytes) distribution over each category.



(c) Average portion of the size of the content loaded in IPv4 and IPv6 with 95% confidence intervals measures, across categories.

Figure 8: Dual-Stack content distribution. The bottom and top whiskers represent the 5th and 95th percentiles, respectively. Outliers are not shown in the boxplots.

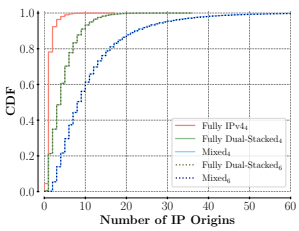
IPv4. Conversely, 35% of the points are positive, showcasing a preference for IPv6, with the remaining 1% balancing out at zero, indicating an equal distribution of resources across both protocols. The figure also includes a count of websites contributing to each bin, providing a clear view of the data distribution and the significant skew towards IPv4.

Fig. 8 delves deeper into the category of content loaded over each protocol. The box plots in Fig. 8a demonstrate that most categories predominantly load over IPv4. *Image* stands out the most by loading only 42% of the images over IPv6 when considering the 95th percentile. However, exceptions are observed with *Fetch & XHR* asynchronous requests and *Font*, which are more frequently loaded over IPv6. Even though the number of *Script* resources appears comparable between IPv4 and IPv6, a significant difference emerges when examining the size of these resources. Fig. 8b reveals that scripts loaded over IPv6 tend to be larger and more complex, with sizes reaching up to 8,000 KB in the 95th percentile, while reaching only the half in IPv4 content.

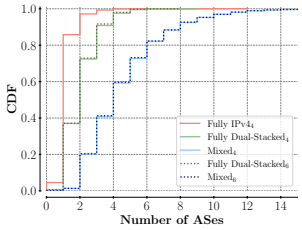
The analysis of average values (Fig. 8c) across these resource categories further enhances our understanding of protocol prioritization by showing that *Media* files also predominantly use IPv4.

4.3 Performance

4.3.1 Parameters Influencing Performance. In Sec. 4.1 we have already considered content-related metrics such as website size and the number of resources, we now integrate network-centric metrics that potentially impact performance: the number of Autonomous Systems (ASes) and the number of IP origins involved during loading. These metrics were derived in the postprocessing phase, using MaxMind geolocation services to perform the IP to AS mapping [38].



(a) Number of ≠ IP origins involved in the loading.



(b) Number of ≠ Autonomous Systems (ASes) involved in the loading.

Figure 9: Performance-related metrics involved in the loading of Fully IPv4, Fully Dual-Stacked, and Mixed websites. Parameters captured during IPv6-preferenced loading are differentiated using subscripts: subscript$_4$ showcases the use of full-IPv4 loading while subscript$_6$ is for IPv6-preferenced loading.

Fig. 9a reveals that Mixed websites tend to load resources from a larger number of different IP addresses (up to 60), while the majority of Fully IPv4 websites (≈ 80%) loads resources from maximum 2 IP addresses. Similarly, Fig. 9b quantifies the amount of ASes involved in loading resources. Mixed websites can rely up to 15 ASes, while Fully IPv4 rely on less than 2 different ASes. It is worth noticing that, despite the different loading strategies, the global distribution of these metrics shows no significant differences. This indicates that the underlying network infrastructure treats full-IPv4 and IPv6-preferenced sessions similarly in terms of routing diversity and connection origination. We have also examined ASes relationships, relying on CAIDA dataset [9], and observed that 80% of the websites analyzed by NetQuartz hosted their main HTML document within an AS that had no direct connections to any other AS involved in the resource

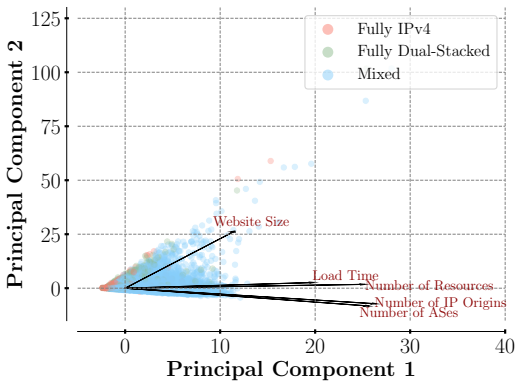


Figure 10: PCA biplot visualizing principal contributions of performance-related metrics such as website size, load time, number of resources; IP origins; ASes- The 2 Principal Components explain 76% of the variance (56% explained by PC1 and 20% explained by PC2).

loading process. This suggests a significant dispersion of resources, whatever the IP protocol in use.

To determine how these metrics interrelate and contribute to website load times, we performed a two-dimensional Principal Component Analysis (PCA). This dimensionality reduction technique helps us visualize the correlations among the variables. The results, depicted in Fig. 10, illustrate that vectors (arrows) with similar orientations are likely to be correlated. We note that the number of resources requested over HTTP/S significantly correlates with load times. Other features such as website size, the number of ASes, and IP origins do not show strong individual correlations with load times. However, when considered together (size + ASes + IP origins), their cumulative impact becomes apparent. These observations suggest that a dual view of both content-related and network-centric metrics is essential to fully understand website performance dynamics.

4.3.2 Website IPv4 *vs.* IPv6 *Load Times.* It has been established in Sec. 4.3.1 that the number of resources significantly impacts load times, with Fully IPv4 websites loading fewer resources compared to Fully Dual-Stacked, and subsequently, Mixed websites. The same relationship can be observed by considering the loading time of the website (Fig. 11a), classes loading fewer resources will therefore load faster.

Regardless of the website's class, IPv6-preferenced sessions outperform the full-IPv4 ones. Mixed class loads 20% faster in median and 11% faster in their 95th percentile when using IPv6-preferenced loading. The difference is even more noticeable in the case of Fully Dual-Stacked class, IPv6-preferenced load 16% faster in median and 24% faster in the 95th percentile. This observation is further pointed out by the boxplots in Figs. 11b, 11c, 11d showing the distribution of loading times across the various content categories. Predominantly, *Image* and *Script* remain the dominant categories, yet in all instances, IPv6-preferenced loading times are slightly better than those of full-IPv4.

Typical justifications for the enhanced performance of IPv6 include the absence of NAT processing, reduced header processing at network nodes, and the avoidance of fragmentation [14, 33]. However, in our case, the argument regarding NAT processing does not hold since our vantage points use publicly addressable IP addresses.

To investigate deeper into potential reasons behind IPv6's performance advantage, we used specific data provided by NetQuartz: the average Round Trip Time (RTT) extracted from the five pings performed to the server hosting the main HTML document (see Sec. 2.1) and the IP addresses logged for each resource during the loading process (see Sec. 2.3).

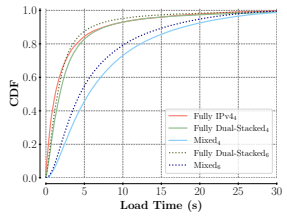
We first examined the RTT estimates, Fig. 12 shows that the distributions between IPv4 and IPv6 for dual-stack servers are the same, so RTT estimation is not an ideal candidate to explain this difference in performance. Nevertheless, there is a clear difference in performance between IPv4-only servers and dual-stack servers, with pings towards dual-stack servers responding 78% faster for the 95th percentile and 71% faster in median.

We then calculated geographical distance – through MaxMind geolocation services [38] – using the Haversine formula to measure the distance between each server hosting a resource and the corresponding vantage point. These distances were weighted by the resource size, scaled down by a factor of 10^6 for readability reasons, producing what we refer to as the "HaverSize score". Assuming we have n resources, each with a resource size s_i (in Kilobytes) and a Haversine distance d_i from the vantage point (in kilometers), and K is the scaling factor set to 10^6, then the HaverSize score of a website is calculated as follow:

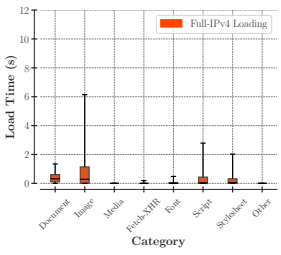
$$\texttt{HaverSize} = \frac{\sum_{i=1}^{n} s_i \times d_i}{K}. \tag{2}$$

The HaverSize score is a non-negative metric (≥ 0), a score close to zero indicates that the sum of the weighted distances for all resources is minimal. Practically, this suggests that most resources are either very small in size or are located very close to the vantage point.

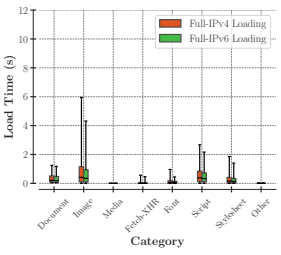
Fig. 13 highlights the distribution of these HaverSize scores, which consistently show a preference for IPv6. This suggests that IPv6-preferenced loading leads to routes that might be more efficient or involve shorter physical distances. It is important to note that IP geolocation and Haversine calculations do not precisely represent the exact network paths taken by IP packets, they still provide useful insights



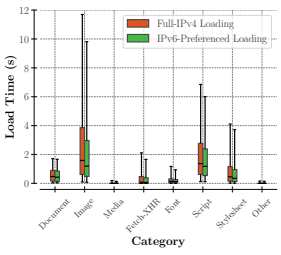
(a) Global load time for `Fully IPv4`, `Fully Dual-Stacked` and `Mixed` websites.



(b) Loading time distribution for `Fully IPv4` class.



(c) Loading time distribution for `Fully Dual-Stacked` class.



(d) Loading time distribution for `Mixed` class.

Figure 11: Website load time distribution (in seconds) of `Fully IPv4`, `Fully Dual-Stacked` and `Mixed` websites. Subscript$_4$ showcases the use of full-`IPv4` loading while subscript$_6$ is for `IPv6`-preferenced loading. The bottom and top whiskers represent the 5th and 95th percentiles, respectively. Outliers are not shown in the boxplots.

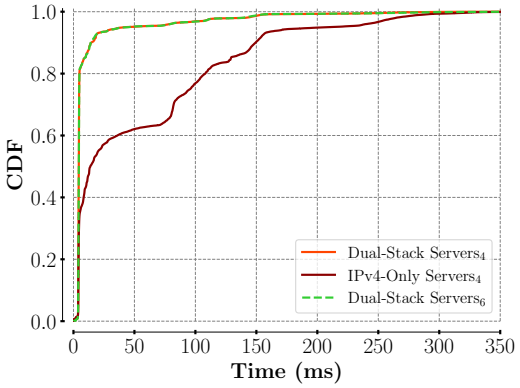


Figure 12: `ping` response times (in milliseconds) for `IPv4` vs. `IPv6` on dual-stacked and `IPv4`-only classes. Subscript$_4$ means that the `ping` is performed over `IPv4` loading while subscript$_6$ is for `IPv6` `pings`.

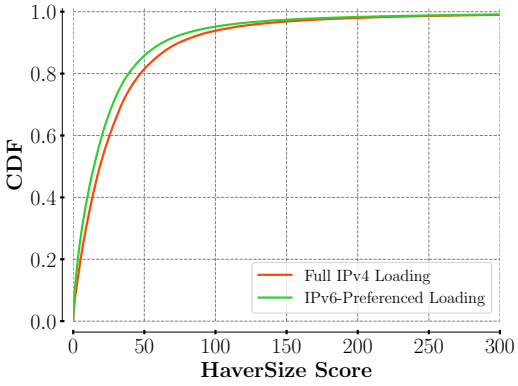


Figure 13: HaverSize scores, where each score is the weighted sum of Haversine distance (in kilometers) between the vantage point and the resource server, with weights proportional to resource sizes, comparing full-`IPv4` and `IPv6`-preferenced loading strategies.

into the underlying website infrastructure and the physical proximity of connected resources.

4.3.3 User Experience. When analyzing resource failure rates (NetQuartz Step 3 – see Sec. 2.3) within `Mixed` websites, we found that 6.21% of these websites encountered one or more read failures when using `IPv4`, and 5.28% experienced similar issues over `IPv6`. While connection failures are less common, they still occur, affecting 1.08% of websites on `IPv4` and 0.98% on `IPv6`. The disparity in failure rates, in favor of `IPv6`, is a direct consequence of the heavier resource demands placed on `IPv4` in these `Mixed` environments (see Sec.4.2). The failures also vary significantly by content type, providing insights into which categories are most susceptible to issues: whatever the website class, *Document* consistently exhibit the highest failure experience accross websites. 66% of the `Fully IPv4` websites are subject to *Document*-related failures, 65% of `Fully Dual-Stacked` and 55% of `Mixed` websites.

As established in Sec. 4.3.1, the number of resources requested via `HTTP/S` significantly influences load time. In light of this observation, many websites are adopting the use of `data URLs` to embed data directly within *Document* resources (NetQuartz Step 3 – see Sec. 2.3). This approach helps to minimize `HTTP` overhead by reducing the number of separate requests required for resource fetching. We examined the usage of `data URLs` across various websites and found that 38% of them incorporate this technology. At the 95th percentile, each website contains up to 8 `data URLs`, embedding a total of 17.2 Kilobyte of data within these `URLs`. We have also observed that the majority of these `data URLs` are concentrated in the *Image* category.

5 RELATED WORK

Recent researchs have evaluated the impact of Happy Eyeballs on websites performance, showing that the default 300ms timer was not always leading to the best outcome [5, 6]. Piraux et al. [41] recently provided a DNS extension as an adaptative address family selection algorithm able to dynamically select, according to end-to-end latency measures, the best address family for establishing the connection.

Research on Internet path-length aspects between IPv4 and IPv6 has also been performed [27]. It showed that IPv6 and IPv4 paths share similar path lengths, implying that, despite IPv6 's less stable paths and higher dynamics, the fundamental infrastructure (in terms of distance data travels) is comparable to IPv4.

Web content has also been examined. Butkiewicz et al. [8] found that a significant portion of web pages includes content fetched from multiple, distinct servers, introducing additional delay. They highlight that it is not just the total size of the page that affects load times but, more significantly, the number of individual objects requested. Other works [8, 34] show that images and JavaScript are among the prominent content types, with JavaScript objects, in particular, contributing substantially to the total page size. Most of the studies were only focused on the root home page of the service.

On the contrary, Lookyloo [5] – a networking tool web interface – focuses on the classification of all the resources linked to the home page along with their category but, to our best knowledge, no paper has been published on it. Bajpai et al. [2, 22] give insights at the content level of dual-stack web servers, their focus was mainly on the failure rate and the performance of content delivery over IPv4 and IPv6. Our work seeks to complement theirs by delving into application-level metrics and evaluating how much of the content is distributed over IPv4 and IPv6.

When discussing web performance, user experience is a crucial factor. Sengupta et al. [44] conducted a study using Google's Chrome User Experience Report (CrUX), which focused specifically on user experience and user-related metrics, particularly those associated with web page rendering. Although this research is orthogonal to our work, it underscores disparities in these metrics across devices and geographic regions, revealing that desktops generally outperform mobile devices. Furthermore, it highlights that certain countries benefit from superior network performance, leading to a better overall user experience.

To date, and to our best knowledge, no studies have explored how web content is distributed across IPv4 and IPv6, along with the associated performance consequences on the different web categories. Our work aims at filling this gap.

6 CONCLUSION

This paper contributes to understanding dual-stack server performance through the development and deployment of NetQuartz, a website data analyzer tool. NetQuartz is meticulously designed to derive and compare the performance of websites across both IPv4 and IPv6 protocols. We successfully collected and analyzed data from 217,974 websites after a two-week measurement campaign.

Our analysis categorizes websites into three distinct classes based on the servers hosting their resources: Fully IPv4, Fully Dual-Stacked, and Mixed. This classification allows us to uncover patterns in web resource management and loading efficiencies. Notably, Fully IPv4 class are generally smaller in size compared to Fully Dual-Stacked class, with Mixed class presenting the largest sizes and greatest number of resources. This observation is particularly insightful as it correlates with our finding that the number of resources is directly linked to loading times.

Our findings also reveal that Fully Dual-Stacked and Mixed classes typically perform better in IPv6-preferenced loading than in full-IPv4 one.

The observation is particularly interesting with Mixed websites, where we find that resources tend to be loaded more over IPv4 than IPv6, yet the overall performance in IPv6-preferenced loading remains more efficient.

Our conclusions could benefit from a broader analysis to better understand the performance differences observed between full-IPv4 and IPv6-preferenced loading. Expanding the number of vantage points and incorporating traceroute analyses could provide a more detailed understanding of the exact paths taken by IPv6 and IPv4 packets.

SOFTWARE ARTEFACT

NetQuartz source code, data analysis scripts, and data collected will be released to the community upon paper acceptance.

ETHICS

The collection of web resources can be a contentious topic, often raising ethical concerns and attracting attention. Scrutinizing web content is generally considered unacceptable, and many web services and hosts have anti-scraping and anti-bot measures in place to discourage such an activity. It is important to note that the focus of this project has been on collecting metadata related to resources, such as size, loading time, or resource category, rather than harvesting their content.

Additionally, we have intentionally avoided using techniques to bypass anti-scraping measures to ensure that no web service is deceived or impersonated, whether by simulating a human user or a web browser. Furthermore, all

inferential rules and heuristics used are exclusively based on information that is either readily available or explicitly provided to the user.

This ethical approach highlights a focus on transparency and respect for web standards and privacy, and avoids invasive or deceptive practices. This project aims at providing valuable insights into web resource dynamics while adhering to ethical guidelines and focusing on metadata. The integrity and security of web entities were not compromised.

REFERENCES

[1] 2024. *MIME Sniffing Standard.* Living Standard. WHATWG: Web Hypertext Application Technology Working Group. https://mimesniff.spec.whatwg.org/ [Last Accessed: February 19th, 2024].

[2] V. Bajpai. 2016. simweb, A tool to measure similarity of webpages delivered over IPv4 and IPv6. https://github.com/steffiejacob/simweb [Last Accessed: March 28th, 2024].

[3] V. Bajpai, J. Ott, and J. Schonwalder. 2015. Measuring Youtube from Dual-Stack Hosts. In *Proc. Passive and Active Measurement Conference (PAM).*

[4] V. Bajpai and J. Schönwälder. 2013. Measuring TCP Connection Establishment Times of Dual-Stacked Web Services. In *Proc. Conference on Network and Service Management (CNSM).*

[5] V. Bajpai and J. Schönwälder. 2016. Measuring the effects of happy eyeballs. In *Proc. ACM/IRTF Applied Networking Research Workshop (ANRW).*

[6] V. Bajpai and J. Schönwälder. 2019. A Longitudinal View of Dual-stacked Websites – Failures, Latency and Happy Eyeballs. *IEEE/ACM Transactions on Networking (ToN)* 27, 2 (April 2019), 577–590.

[7] F. Brockners, S. Bhandari, and T. Mizrahi. 2022. *Data Fields for In-Situ Operations, Administration, and Maintenance(IOAM).* RFC 9197. Internet Engineering Task Force.

[8] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. 2011. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proc. ACM Internet Measurement Conference (IMC).*

[9] CAIDA. 2024. *AS Relationships.* https://www.caida.org/catalog/datasets/as-relationships/ [Last Accessed: April, 18th 2024].

[10] B. Carpenter and S. Jiang. 2013. *Transmission and Processing of IPv6 Extension Headers.* RFC 7045. Internet Engineering Task Force.

[11] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. 2010. Evaluating IPv6 adoption in the Internet. In *Proc. Passive and Active Measurement Conference (PAM).*

[12] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey. 2014. Measuring IPv6 adoption. In *Proc. ACM SIGCOMM.*

[13] DARPA. 1981. *Internet Protocol.* RFC 791. Internet Engineering Task Force.

[14] J. Davies. 2012. *Understanding IPv6* (3rd ed.). Microsoft Press. 91–115 pages. [Chapter 4: The IPv6 Header].

[15] S. Deering and R. Hinden. 1998. *Internet Protocol, Version 6 (IPv6).* RFC 2460. Internet Engineering Task Force.

[16] S. Deering and R. Hinden. 2017. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200. Internet Engineering Task Force.

[17] A. Dhamdhere, M. Luckie, B. Huffaker, kc claffy, A. Elmokashfi, and E. Aben. 2012. Measuring the deployment of IPv6: topology, routing and performance. In *Proc. ACM Internet Measurement Conference (IMC).*

[18] DigitalOcean. 2024. Cloud Infrastructure for Developers. https://www.digitalocean.com [Last Accessed: April 28th, 2024].

[19] Chromium Distribution. 2024. Webdrivers for Chrome (download page). https://chromedriver.chromium.org/downloads.

[20] DomCop. 2023. Top 10 million Websites Based on Open PageRank data. https://www.domcop.com/top-10-million-websites [Last Accessed: December 19th, 2023].

[21] D. Eastlake, Smith C., and D. Soroka. 2000. *HTTP MIME Type Handler Detection.* RFC 2936. Internet Engineering Task Force.

[22] S. J. Eravuchira, V. Bajpai, J. Schönwälder, and S. Crawford. 2016. Measuring Web Similarity from Dual-stacked Hosts. In *Proc. International Conference on Network and Service Management (CNSM).*

[23] e. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner. 2014. Web Data Extraction, Applications and Techniques: A Survey. *Knowledge-Based Systems* 70 (November 2014), 301–323.

[24] R. Fielding, M. Nottingham, and J. Reschke. 2022. *HTTP Semantics.* RFC 9110. Internet Engineering Task Force.

[25] R. Fielding, M. Nottingham, and J. Reschke. 2022. *Hypertext Transfer Protocol (HTTP/1.1).* RFC 9112. Internet Engineering Task Force.

[26] C. Filsfils, S. Previdi, L. Grinsberg, B. Decraene, S. Likowski, and R. Shakir. 2018. *Segment Routing Architecture.* RFC 8402. Internet Engineering Task Force.

[27] F. Golkar, T. Dreibholz, and A. Kvalbein. 2014. Measuring and Comparing Internet Path Stability in IPv4 and IPv6. In *Proc. International Conference and Workshop on the Network of the Future (NOF).*

[28] Google. 2024. IPv6 deployment statistics. (2024). https://www.google.com/intl/en/ipv6/statistics.html [Last Accessed: February 27th, 2024].

[29] HTTPArchive. 2012-2024. Report: State of the Web, Total Requests. https://httparchive.org/reports/state-of-the-web#reqTotal [Last Accessed: May 8th, 2024].

[30] G. Huston. 2013–2024. *IPv4 Address Report.* https://ipv4.potaroo.net [Last Accessed: May, 8th 2024].

[31] G. Huston. 2015. Examining IPv6 Performance. https://labs.ripe.net/author/gih/examining-ipv6-performance/ [Last Accessed: February 17th, 2024].

[32] G. Huston. 2015. Is IPv6 Faster. https://ripe71.ripe.net/archives/video/1219/ [Last Accessed: March 17th, 2024].

[33] G. Huston. 2016. Evaluating IPv4 and IPv6 Packet Fragmentation. https://labs.ripe.net/author/gih/evaluating-ipv4-and-ipv6-packet-fragmentation/ [Last Accessed: May 13th, 2024].

[34] S. Ihm and V. S. Pai. 2011. Towards Understanding Modern Web Traffic. In *Proc. ACM Internet Measurement Conference (IMC).*

[35] Juniper. 2023. *Understanding Dual Stacking of IPv4 and IPv6 Unicast Addresses.* https://www.juniper.net/documentation/us/en/software/junos/is-is/topics/concept/ipv6-dual-stack-understanding.html [Last Accessed: May, 8th 2024].

[36] G. Kacmarcik and T. Leithead. 2024. *UI events.* W3C Working Draft. W3C: The World Wide Web Consortium. https://www.w3.org/TR/uievents/#event-type-load [Last Accessed: March 19th, 2024].

[37] L. Masinter. 1998. *The data URL scheme.* RFC 2397. Internet Engineering Task Force.

[38] MaxMind. 2024. *Localize content with GeoIP.* https://www.maxmind.com/ [Last Accessed: May 4th, 2024].

[39] M. Nikkhah and R. Guérin. 2015. Migrating the Internet to IPv6: an exploration of the when and why. *IEEE/ACM Transactions on Networking (ToN)* 24, 4 (August 2015), 2291 – 2304.

[40] OVHcloud France. 2024. : Cloud Computing and Web Hosting. https://www.ovhcloud.com/ [Last Accessed: April 28th, 2024].

[41] M. Piraux and O. Bonaventure. 2023. *Adaptive Address Family Selection for Latency-Sensitive Applications on Dual-stack Hosts.* cs.NI 2309.05369. arXiv.

[42] D. Schinazi and T. Pauly. 2017. *Happy Eyeballs Version 2: Better Connectivity Using Concurrency.* RFC 8305. Internet Engineering Task Force.

[43] Selenium. 2023. Selenium automates browsers. That's it! https://www.selenium.dev [Last Accessed: February 19th, 2024].

[44] J. Sengupta, T. Shreedhar, D. Nguyen, R. Kramer, and V. Bajpai. 2024. *Through the Lens of Google CrUX: Dissecting Web Browsing Experience Across Devices and Countries.* cs.NI 2308.06409. arXiv.

[45] S. Stewart and D. Burns. 2024. *WebDriver Protocol Standard.* W3C Working Draft. W3C: The World Wide Web Consortium. https://www.w3.org/TR/webdriver2/ [Last Accessed: March 17th, 2024].

[46] D. Thaler, R. P. Draves, A. Matsumoto, and T. Chown. 2012. *Default Address Selection for Internet Protocol Version 6 (IPv6).* RFC 6724. Internet Engineering Task Force.

[47] A. van Kesteren. 2024. *HTML Standard.* Living Standard. WHATWG: Web Hypertext Application Technology Working Group. https://html.spec.whatwg.org/multipage/ [Last Accessed : March 17th, 2024].

[48] A. van Kesteren. 2024. *URL Standard.* Living Standard. WHATWG: Web Hypertext Application Technology Working Group. https://url.spec.whatwg.org [Last Accessed: March 17th, 2024].

[49] VULTR. 2024. The Everywhere Cloud. https://www.vultr.com/ [Last Accessed: April 28th, 2024].

[50] D. Wing and A. Yourtchenko. 2012. *Happy Eyeballs: Success with Dual-Stack Hosts.* RFC 6555. Internet Engineering Task Force.

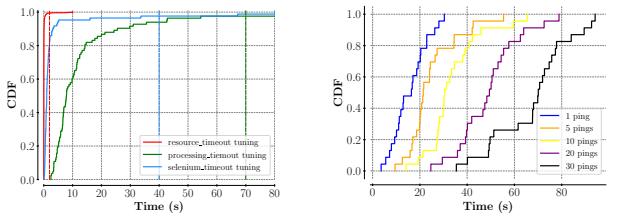
A NETQUARTZ HYPERPARAMETERS TUNING

NetQuartz timeout-hyperparameters (Sec. 2.4) values are fixed based on Cumulative Distribution Functions (CDFs) illustrated in Fig. 14a to determine the most effective timeout settings for the different stages of processing. The tuning is based on 100 websites randomly sampled from the top 300,000 websites in `DomCop`'s list. `selenium_timeout` (with x-axis being the time spent loading the website with selenium) is fixed to 40 seconds, to capture 95% of the websites from the tested sample. Similarly, `processing_timeout` (with x-axis being the total time spent by NetQuartz to process the website) is fixed to 70 seconds. `resource_timeout` (with x-axis being the time spent loading the web resource) is intentionally set to 2 seconds to capture 100% of resources, the goal is to prevent resources from loading indifinitely, we still want to capture a maximum amount of them. The number of `IPv4`– and `IPv6` when applicable – `pings` is set to 5. Fig. 14b shows that with 5 pings, we still get a reliable snapshot of the network's conditions without significantly increasing the overall processing time per domain, ensuring that NetQuartz remains efficient in processing the top300,000 websites from `DomCop`.

B HEURISTICS

B.1 Reclassification of *Fetch*, *XHR* and *Other* Resources

Our heuristic, based on `HTTP` *content-type* header analysis 3.4.1, is validated against `Chrome`'s network performance tool classification which employs requisite techniques for optimal category derivation. The heuristic's foundation lies in keyword detection within the header encoding the resource media type as highlighted in Fig. 3.



(a) Tuning of the timer values. Dotted lines highlight the chosen timer value.

(b) Tuning of the number of pings to perform, x-axis is the total processing time spent on a website by NetQuartz.

Figure 14: NetQuartz hyperparameter tuning.

Type	True Positive		False Positive		True Negative		False Negative		Accuracy	Recall	f1-score
	Raw	Prop.	Raw	Prop.	Raw	Prop.	Raw	Prop.			
Document	31,623	0.033	16,290	0.017	888,298	0.947	960	0.001	0.66	0.97	0.785
Image	367,260	0.391	1,970	0.002	540,339	0.576	27,602	0.029	0.994	0.93	0.961
Media	2,799	0.002	47	0.000	933,813	0.996	512	0.001	0.983	0.845	0.91
Stylesheet	101,775	0.108	929	0.001	833,066	0.888	1,401	0.001	0.99	0.986	0.988
Script	332,304	0.354	8,644	0.009	589,039	0.628	7,184	0.007	0.974	0.978	0.976
Font	53,385	0.056	520	0.001	872,900	0.931	10,366	0.011	0.99	0.837	0.907

Table 4: Content classification heuristic evaluation (total: 937,171 resources) with advanced metrics.

Validation of this heuristic implied a dataset of 25,000 websites, encompassing 937,171 resources, and involved confusion matrix construction to evaluate classification performance. Results, including precision, recall, and f1-score for each category, are detailed in Table 4. The f1-score, representing the harmonic mean of precision and recall, shows robust outcomes across categories, with scores exceeding 90% except for the Document category, which faced a precision challenge at 66%. This anomaly comes from instances where resources, like images, fail to load correctly (e.g., returning an `HTML` document with status code other than 200), yet are classified as Images by `Chrome`'s performance tool. This discrepancy raises questions about `Chrome`'s internal decision-making processes, which are not publicly documented.