2013-ENST-0017

EDITE - ED 130

Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « INFORMATIQUE et RESEAUX »

présentée et soutenue publiquement par

Heng CUI

le 11 avril 2013

Analyse et Diagnostic des Performances Web du Point de Vue de l'Utilisateur

Directeur de thèse : **Ernst BIERSACK**

Jury

M. Refik MOLVA, EURECOM, Sophia Antipolis - France	Examinateur et Président
M. Chadi BARAKAT, INRIA, Sophia Antipolis - France	Rapporteur
M. Guillaume URVOY-KELLER, Laboratoire I3S, Sophia Antipolis - France	Rapporteur
M. Antonio PESCAPÈ, University of Napoli "Federico II", Napoli - Italie	Examinateur
M. Matti Mikael SIEKKINEN, AALTO University, Aalto - Finlande	Examinateur

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

THÈSE

Abstract

In recent years, the interest of the research community in the performance of Web browsing has grown steadily. Due to the rapid evolution of Internet and Web technologies, the needs for understanding the Web performance never end. In order to reveal end-user perceived performance of Web browsing, researchers started to move their measurements “up the stack” and to study real user experiences. In this thesis work, we address multiple issues of Web browsing performance from the perspective of the end-user.

The thesis is composed by three parts: performance analysis, performance diagnosis and a new methodology for critical path analysis. The first part introduces our initial platform which is based on browser-level measurements. We explain measurement metrics that can be easily acquired from the browser and indicators for end-user experience. Then, we use clustering techniques to correlate higher-level performance metrics with lower level metrics. As we will show, such clustering methods can be used in different scenarios to study the relationship between QoE and QoS, and compare and explain the performances experienced in different homes.

In the second part, we present our diagnosis tool called FireLog. We first discuss different possible causes that can prevent a Web page to achieve fast rendering; then, we describe details of the tool’s components and its measurements. Based on the measured metrics, we illustrate our model for the performance diagnosis in an automatic fashion. Finally, we use both controlled experiments and long-term wild deployments to evaluate FireLog.

In the last part, we propose a new methodology named Critical Path Method for the Web performance analysis. We first explain details about Web browser’s intrinsic features during page rendering and then we formalize our the methodology. By adopting this critical path method, we transform Web page rendering process into a Directed Acyclic Graph. With help of the assigned timings, we can derive a final critical path that has a direct impact on the overall Web page rendering performance.

Résumé

Au cours des dernières années, l'intérêt porté aux performances de la navigation internet n'a cessé de croître au sein de la communauté scientifique. En raison de l'évolution rapide des technologies de l'Internet et du Web, la compréhension des performances de la navigation Web est un sujet très riche. Afin de comprendre la perception qu'a l'utilisateur des performances de la navigation Web, les chercheurs ont commencé à déplacer leurs mesures "up the stack" et à étudier l'expérience réellement vécue par l'utilisateur. Au cours de cette thèse nous abordons différents problèmes liés aux performances de lanavigation Internet telle qu'elle est perçue par l'utilisateur final.

Cette thèse se compose de trois parties: l'analyse des performances, le diagnostic des performances et une nouvelle méthodologie pour l'analyse du chemin critique. La première partie présente notre nouvelle plateforme basée sur des mesures faites au niveau du navigateur. Nous présentons les différents paramètres que nous obtenons facilement à partir du navigateur, ainsi que des indicateurs du ressenti de l'utilisateur final. Ensuite nous utilisons des techniques de partitionnement de données afin de trouver les corrélations existantes entre performancesde haut niveau et de bas niveau. Présentées par la suite, ces méthodes de classifications peuvent être utilisées dans différents scénarios pour étudier la corrélation entre QoE et QoS ainsi expliquerles différentes performances qui peuvent être expérimentées au sein de différents foyers.

Dans une seconde partie, nous présentons notre outil de diagnostic appelé "Firelog". Nous étudions tout d'abord les différentes causes qui peuvent affecter le rendu d'une page Web. Ensuite nous décrivons en détails les différents composant de notre outil et les mesures qu'il effectue. Sur la base des paramètres mesurés, nous illustrons notre modèle pour le diagnostic des performances d'une manière automatique. Dans un dernier temps, nous testons Firelog dans un milieu contrôlé et en situation réelle.

Dans la dernière partie, nous proposons une nouvelle méthodologie, "Critical Path Method" (ou Méthode du Chemin Critique) pour l'analyse des performances de la navigation Web. Nous expliquons d'abord en détails les caractéristiques intrinsèques du Navigateur lors du rendu d'une page Web, puis nous présentons formellement notre méthodologie. En adoptant cette Méthode du Chemin Critique, nous traduisons le processus de rendu d'une page Web en un graphe acyclique orienté (DAG). Avec l'aide des timings assignés, nous parvenons à construire un chemin critique, reflétant l'impact direct des différentes métriques sur le rendu global de la page Web.

Acknowledgment

As the same with most Ph.D students, the first person that I would like to thank is my thesis supervisor Prof. Dr. Ernst W. Biersack. I would say that I am really lucky to have the opportunity to work with him during these years. He is a very good guide and has rich experiences on my Ph.D supervision. Each time when my thesis work was going to a wrong direction, he could always discover it in time and lead me in the correct direction. Most importantly, he gave me valuable attitudes to do the research work which are already transformed into a scientific habit of mine.

Special thanks also give to all jury members of my thesis defense, Dr. Chadi Barakat, Dr. Refik Molva, Dr. Antonio Pescapè, Dr. Matti Siekkinen and Dr. Guillaume Urvoy-Keller. We have many interesting discussions during my defense and they also provide lots of constructive comments on my work, not only for this Ph.D thesis, but also different directions for the future work.

I am also very appreciated to work in Eurecom these years since I have made lots of friends here. All the colleagues are very kind and willing to help whenever I have problems. I would like to thank Hadrien Hours, Miriam Redi, Louis Plissonneau, Giuseppe Reina, Aurélien Francillon, Marco Milanesio and all the other colleagues in Eurecom for sharing their knowledge and discussions. Of course, I never forget all the Chinese friends in Eurecom and I will always remember the time that we have spent together. Thank you for all of you.

I would like to particularly thank Chen Jinbang who have contributed lots of effort on my thesis work. Each time when I have problems for either technical or non-technical issues, the first person in my mind for asking help is always him. Without his help and contribution, many experiments in this thesis are definitely impossible to achieve and some interesting results can not be observed neither.

I am also very lucky that I have the chance to work with many excellent students these years. I would like to thank Tomasz Grubba, Chen Yulu, Ye Mintao, Salvatore Balzano. Without their contributions, Chapter 4 of this thesis is impossible to be done.

Moreover, I would like to thank all the IT service, secretary, HR and administrative staffs, especially Gwenaëlle Le Stir and Chrisitine Roussel. They are really professional and always friendly, and their help save me a lot of time on the bureaucratic or lab issues etc.

Last but not the least thanks are to my family. I know that I owe my parents so much and they always support my career and keep encouraging me no matter how depressed I am. I am very proud to be the son of them. Deep thanks also go to my lovely girl friend Deng Mingzhu, who is the only one that spent all these years together with me and always stands on my side.

Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

"*You affect the world by what you browse.*"

Tim Berners-Lee

1.1 Motivation

The World Wide Web (also known as **WWW** or **the Web**) is a global system of hypertext documents viewed by a Web browser. It was invented by Tim Berners-Lee in 1989 at CERN. The year after, he also built the first Web browser [30] and Web server [32].

Although it has been only 20 years since the Web is invented, technologies built on top of it are growing at a tremendous rate. A report in [60] declares that from 1995 to 2003, Web traffic is the largest portion of Internet traffic. Similar observation also appears in [8] which finds the Web traffic has already overtaken Peer-to-Peer (P2P) traffic and "*consumed the largest percentage of bandwidth on the network*". For the end user, Web related applications also become the most popular usage for the Internet surfing [79]. Examples for Web usages are consulting a Wikipedia entry, accessing a news page, on-line shopping, or viewing user generated content such as YouTube or Dailymotion.

Plenty of researches or surveys have already shown that the poor performance related to the Web can have great impacts both for companies and the end users. Google, as the most popular Web service provider, makes great efforts to improve the performances [12]. An experiment done by Google [42] clearly shows that with the latency increasing from 100 to 400 ms, their daily number of searches per user will be reduced by 0.2% to 0.6%. Another Gomez white paper [11] shows similar conclusions: for a $\$100,000$/day e-commerce Web site, 1-second more delay means 2% customer lost and $\$2.5$ million reduction in the yearly revenue. `Amazon.com` [71] also reports that every 100 ms increase in the page load time can decrease their sales by 1%.

On the other hand, from the end user's point of view, they can easily become impatient or abandon their current page when the page is slow (e.g. [11] [53] [62] [75]), or even switch to a Web competitor ([10] [43]).

Based on the above discussions, we need diagnostic systems or methodologies that help to analyze the Web performance for the end user, such tools can help in identifying the performance bottlenecks for their browsed Web pages; while for the service providers, it can be also helpful to quantify the

Quality of Experiences (QoE) for their own customers. Being motivated by these issues, in this thesis, we focus on developing tools or methodologies for the Web browsing performances.

1.2 Background Knowledge

As we know that given a Web page, it may contain multiple types of objects. Such objects can be the main skeleton like basic `HTML` file, and also others like image files or cascading style sheets etc. With the growth of multimedia technologies, recent Web pages [64] also contain more multimedia objects such as flash video etc. To fill the Web page, the browser first downloads all the necessary objects inside the page, then render them on the corresponding positions. During the downloading phase, several protocols in different levels are mainly involved.

Upper Layer Traditionally, HTTP (Hypertext Transfer Protocol) is used as the application level protocol[1] for communications between end user and the Web servers. Such protocol is mainly based on a "request-response" model, where requests are normally from the client and Web server provides the corresponding response. Fig. 1.1 shows an example of the HTTP header for downloading a single object[2] from a Web page. Fig. 1.1(a) shows the HTTP request headers and Fig. 1.1(b) shows the response ones. Here, we briefly discuss some headers that may relate to this thesis work while other details can be referred in [21].

- `GET`: At the first line of HTTP request, the client needs to indicate the needed object to the server by a GET method.
- `Host`: This header indicates to the browser which Web server is responsible for the object. If the host name is given not in the form of a standard IP address, before the HTTP request, it needs a DNS request for the name resolving.
- `Connection`: Such header with value of `keep-alive` means that persistent connections can be used for next request. `keep-alive` is used as the default value since HTTP/1.1, however, if such `Connection` header has the value `close`, persistent connections will not be used.
- `Referer`: This header identifies the source URI (Uniform Resource Identifier) of given download object, which is the main `HTML` object in our example. It's also useful to query Web page that given object belonging to [64].
- `Response Code`: As the first message in the HTTP response, the server needs to clarify whether the requested object exist or not. In our case, `200` indicates existence of the objects while other common response codes can be `3xx` or `4xx`, indicating redirection or errors respectively.
- `Content-Length`: This header value is set by the Web server to indicate size of the object. However, as is shown in [83], there are mis-matching between the declared value and real object size. Therefore, in this thesis, we will use the real received bytes instead of the indicated ones by this header.

Besides the HTTP protocol, DNS (Domain Name System) is also used for the name resolving.

[1]Recent years, HTTPS (Hypertext Transfer Protocol Secure), which entirely piggybacks HTTP protocol, is also widely used for Web page downloading.

[2]Downloading object `www.bing.com/fd/s/a/hpc3.png` from `www.bing.com` Web page.

```
GET /fd/s/a/hpc3.png HTTP/1.1
Host:  www.bing.com
User-Agent:  Mozilla/5.0 (X11; Linux ....
Accept:  image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language:  en-us,en;q=0.5
Accept-Encoding:  gzip, deflate
Connection:  keep-alive
Referer:  http://www.bing.com/
Cookie:  _FP=EM=1; _FS=NU=1; _SS=SID=....
```

(a) HTTP **Request** Header

```
HTTP/1.1 200 OK
Content-Type:  image/png
Last-Modified:  Tue, 14 Aug 2012 20:38:35 GMT
Content-Length:  5448
X-N: S
Cache-Control:  public, max-age=15535305
Date:  Fri, 04 Jan 2013 09:47:48 GMT
Connection:  keep-alive
```

(b) HTTP **Response** Header

Figure 1.1: HTTP Headers for Downloading One Object Using Firefox Browser

Lower Layer As lower layer protocols, TCP (Transmission Control Protocol) is used for HTTP transactions and DNS is transferred on top of UDP (User Datagram Protocol). TCP is close-loop and provides reliable end-to-end transmissions using Sequence and Acknowledgment numbers. Handshake phase is also needed for TCP to establish the channel, congestion control and loss recovery for TCP are also well-standardized [40] [39] [86] [76]. UDP only provides best-effort transmissions and does not guarantee a reliable transmission channel, in this case, DNS loss can only be recovered by retransmission control in the higher layer.

Arguments Although these protocols used in downloading Web pages have evolved over ten years, arguments about their limitations still exist. Among these arguments, most of them lay on the interactions between the higher and lower layer protocols, for example:

1. Overhead of TCP in terms of the initial Round Trip Time (RTT) in the handshake phase is considered less efficient. In this case, Radhakrishnan et al. in [78] proposed `TCP Fast Open`, an extension for current TCP protocol, which allows the data transmission during handshake phase. This proposal then, gets positive support in the industrial area [48] [28] [29] as well.

2. TCP Initial Congestion Window (ICW) is another feature that is commonly argued. Since Web page objects are normally small in the bytes [51] [64], traditional ICW value (4KB defined in [38]) is sometimes considered not progressive enough for some domains. Dukkipati et al. in [51] even argue to increase the ICW up to 10*MSS (Maximum Segment Size). Currently, some content providers [19] have also tuned their ICW much higher than the traditionally defined 4KB.

3. Besides previous TCP settings that may affect HTTP transactions, another argument relies on the loss recovery by TCP. For example, Paxson et al. [76] update the TCP specification to decrease the first Retransmission Time Out value from 3 seconds to 1 second.

There are also other comments for revising higher application layer protocols [27] [13]. In this thesis, we will not focus on the protocol modifications and still use traditional standardized ones for the Web diagnosis and analysis.

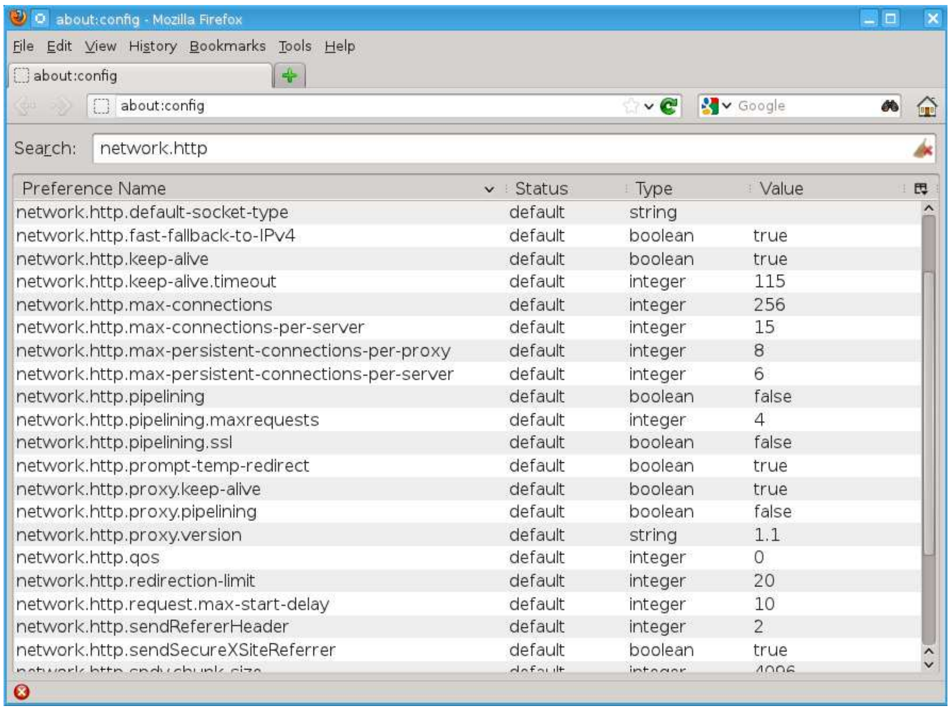
1.3 Challenges of Our Study

There are some challenges that we will face in this thesis work. To sum them up, we categorize them into the following groups:

1. The first type of challenge is due to the diversity of connections. As we have seen previously multiple types of connections are involved such as TCP or DNS (UDP) connections. Performance studies of each individual connection are well investigated in the literature (e.g. [36] [85] [58] etc.). However, few of these studies have taken all those connections into account for the whole Web page. This can be due to many reasons, such as:
    a) such grouping may be tricky and needs more information than traditional TCP/IP headers which makes the packet capturing much heavier;
    b) dependencies between each object inside a Web page always exist, therefore, it makes the study of different connections' impact harder.
2. The second type of challenge is due to the fact that our goal of analysis and diagnosis is to serve for real users, therefore, we need also to take the impact on real user into account. Such impacts can be considered as twofold:
    a) there is no unique objective measured metrics that can exactly represent user's real satisfaction on their browsed Web page, unless using the real subjective feedback. In this case, biases will always exist for providing such feedback [61] [82];
    b) even for the measured objective metrics, they can be also affected by user's random behaviors. For example, browsing a Web page from an empty page or linked pages may lead to different captured metrics: the former case needs to establish new connections while the latter case might re-use persistent connections where no TCP handshake delay is observed.
3. The third important challenge comes from the Web browser itself, and this factor is also important for the Web performance:
    a) there are huge amount of configuration parameters inside the Web browser and each of them may potentially affect the final browsing experiences. As an example, Fig. 1.2 shows a snapshot of Firefox browser's configurations (`about:config` page) for HTTP query. We can see that there are many parameters that may affect browser rendering actions such as parallel connections, proxy set-up, redirect limitation, pipelining etc. Considering only parallel connections, there are already four different parameters[3]. Moreover, such settings can become more complicated if we take other browsers such as IE (Internet Explorer) into account [3];

[3]Parameter names: `network.http.max-connections`, `network.http.max-connections-per-server`, `network.http.max-persistent-connections-per-proxy`, `network.http.max-persistent-connections-per-server`.

b) the Web browser can by itself still impact user perceived performance a lot. For instance, IE may block Web page rendering due to downloading of `stylesheet` objects, user may perceive completely diverse performances under different stylesheet positions in the HTML script even for the same Web page[4]. Similar blocking also exists for the `script` objects for both IE and Firefox[5].


about:config - Mozilla Firefox
File Edit View History Bookmarks Tools Help
about:config
about:config
Google

Search: network.http

Preference Name	Status	Type	Value
network.http.default-socket-type	default	string	
network.http.fast-fallback-to-IPv4	default	boolean	true
network.http.keep-alive	default	boolean	true
network.http.keep-alive.timeout	default	integer	115
network.http.max-connections	default	integer	256
network.http.max-connections-per-server	default	integer	15
network.http.max-persistent-connections-per-proxy	default	integer	8
network.http.max-persistent-connections-per-server	default	integer	6
network.http.pipelining	default	boolean	false
network.http.pipelining.maxrequests	default	integer	4
network.http.pipelining.ssl	default	boolean	false
network.http.prompt-temp-redirect	default	boolean	true
network.http.proxy.keep-alive	default	boolean	true
network.http.proxy.pipelining	default	boolean	false
network.http.proxy.version	default	string	1.1
network.http.qos	default	integer	0
network.http.redirection-limit	default	integer	20
network.http.request.max-start-delay	default	integer	10
network.http.sendRefererHeader	default	integer	2
network.http.sendSecureXSiteReferrer	default	boolean	true

Figure 1.2: Snapshot of Firefox (version 12.0) Configurations on HTTP Query

4. The other challenge that we are facing in this thesis is the Web performance diagnosis. Due to large range of areas that the thesis work may link with, therefore, researchers from different areas may draw completely different conclusions for the root cause of Web performances. In this thesis work, we will focus more on the latency factors in the downloading of objects while ignoring the browser specific features such as execution speed etc.

1.4 Thesis Claim

In this thesis, we make the following claims:

I Our initial platform which combines measurements at both higher user's level and lower network level can greatly reduce the gap between user's *subjective* QoE (Quality of Experience) and *objectively* measured QoS (Quality of Service).

II Different kinds of clustering techniques can be adopted to study the Web performance:

[4]Two examples of Web pages with the same objects, `css` at the bottom: http://stevesouders.com/examples/css-bottom.php and `css` at the top: http://stevesouders.com/examples/css-top.php

[5]An example of scripts at the top v.s. bottom: http://stevesouders.com/examples/move-scripts.php

- Using traditional centralized clustering helps us to reveal the relationship between user level satisfaction (QoE) and lower network level (QoS) metrics.
- Distributed clustering provides scalable approach to use the measurements performed of different users.

III We propose root cause analysis model based on passive measurements from user's wild browsing. Our method is implemented via the Mozilla development interfaces and stores the data in a database.

IV Sharing browsing experiences help us develop a novel approach in the analysis of the Web page browsing. Our novel approach named Critical Path Method can be useful for the Web performance analysis in several scenarios including providing *what-if* predictions and key object extraction.

1.5 Thesis Organization

The structure of the thesis follows largely the claims and is divided into three parts:

In the first part, we propose the initial architecture (Chapter 2) for evaluating the Web performance and present our idea of browser-based methodology for the analysis. Based on this initial prototype, we use clustering methodology, both in centralized (Chapter 3) and distributed (Chapter 4) manner, to analyze and diagnose poor performing Web browsings. These discussions address claim I and II of previous section.

In the second part, we address claim III of the thesis. We first describe the adaption of our tool for wild home deployments (Chapter 5); and then, discuss the performance limitation scenarios that we focus on; finally, we develop an automated model for the performance diagnosis (Chapter 6).

In the third part, we propose a new methodology to analyze the Web. Since discussions in previous chapters take all objects within a Web page as a whole entity and do not distinguish the importance among them. In this case, we focus on finding critical events during a whole page rendering and addresses claim IV.

Finally, in the conclusion chapter (Chapter 8), we revisit all the thesis claims and evaluate the thesis. We also propose several different directions for the future work based on these years working experiences on this topic.

Part I

Web Performance Analysis Based on User Experiences

Overview of Part I

In Part I, we describe and justify our methodology for analyzing Web browsing performance, specifically for the end user perceived performance.

In Chapter 2, we introduce our initial set-up for measuring the Web. We describe how it works at different layers and how we combine the measurements together. We then define some metrics from the platform and perform several case studies to illustrate its use.

In Chapter 3, we further extend our initial platform and add user's interactions. We then propose our methodology to study the relationship between subjective user's QoE (Quality of Experiences) and objective measured QoS (Quality of Service) metrics. By adopting the methodology on the collected data from real users, we demonstrate the potential of the proposed methodology, not only for uncovering the relationships, but also for discovering different features across multiple homes.

In Chapter 4, we extend previous methodology of Chapter 3 into a distributed fashion. We describe general idea of the distributed method and perform experiments from multiple homes. Our experiments reveal that its results can be considered as the approximation to the centralized method. Moreover, we also show that although the distributed method can only provide limited information for the final results (i.e. only centroids and counters are available for each client), we can still illustrate different features among multiple homes.

In summary, this part essentially describes the basic idea of our measurement methodology for the Web and use the "**analysis**" way to diagnose poor performance for end users.

CHAPTER 2
Initial Platform

"The only way to know how customers see your business is to look at it through their eyes."

Daniel R. Scroggin

2.1 Introduction

In Web browsing, a low response time is key for good user experience. Low level metrics such as packet loss or round trip times are not able to measure user experience, but may be useful to explain performance problems. For this reason, in this chapter, we propose our initial platform that puts an *eye* on user's screen while browsing the Web page and combines information obtained from a Web browser plugin with a low level packet capture system.

While the measurements at browser level allow us to *detect* problems, the measurements at packet level allow us to *explain* problems if they are caused by network elements along the path from the client to the server or by the server itself.

2.2 Platform Set-up

As shown in Fig. 2.1, our platform setup is composed by three parts: a plugin for the Web browser; a packet-level capture and a database repository (DBMS).

2.2.1 Browser Plugin

We use Firefox as our web browser due to its rich APIs [23] for development. The browser plugin tracks some critical events during a Web session that are related to the user perceived experience. We track paint events of the browser to measure the **waiting time** that elapses between the user clicking on a Web page and the content of the Web page being displayed. We use the plugin to add event-listeners inside the browser:

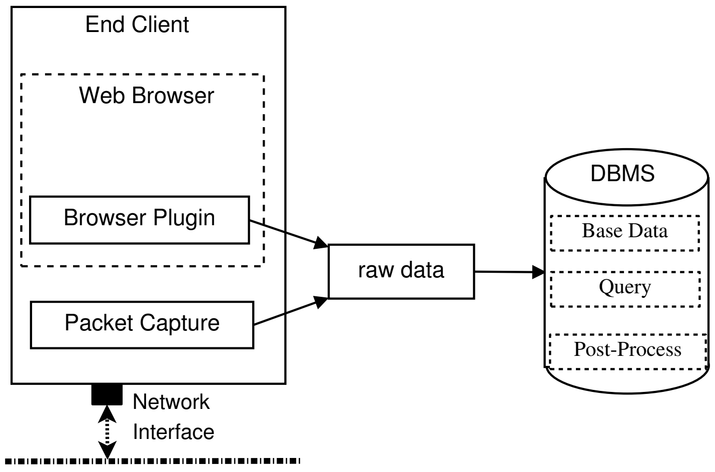


Figure 2.1: Initial Platform Architecture

- The **first painting event** by the browser tells how long the user needs to wait for the *first visual impact*. We call the time interval between the user clicking on the URL until the first painting event the **first impression time**. A large first impression time means that the user has to wait for a long time until he sees anything of the Web page, which will result in a poor user experience.

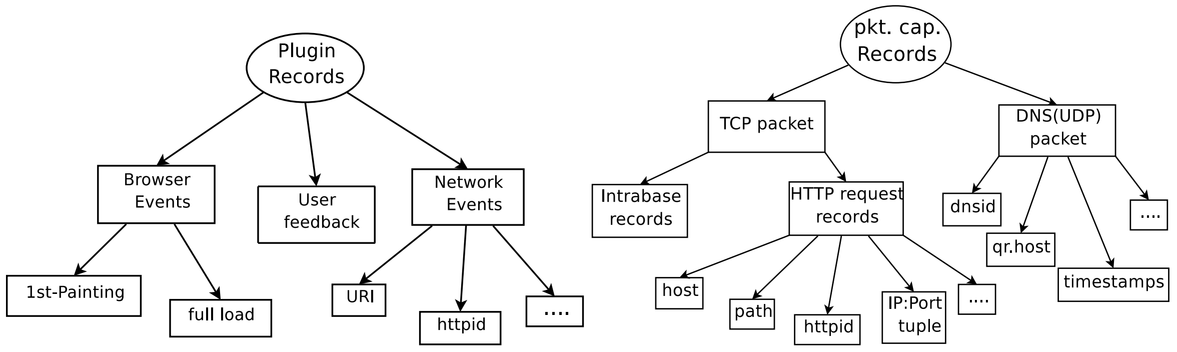
- The **full page load event** measures how long it takes until the entire content of a Web page, which may consist of several tens of different objects is displayed. We call the time interval between the user clicking on the URL until the full page load event the **full load time** T_{full}.

We use the term **waiting time** to refer to either the first impression time or the full load time and we use the term **web session** to the time interval between the user clicking on a URL and the requested Web page being fully loaded. For each Web object that is loaded, the plugin records a certain number of additional information such as object's URI (Uniform Resource Identifier) or httpid (explain later) etc.

2.2.2 Packet Level Capture

We perform a network level capture using wireshark or tcpdump to record all the data packets exchanged during a Web session. For convenience we write all the raw data packets into a database management system. We use PostgreSQL [26] as our DBMS, the PL/PgSQL as its programming language, and the Intrabase system shown in Fig. 2.2(b) for TCP connection analysis is developed by Siekkinen [84].

Since the plugin and packet capture are working simultaneously but independently, we need a way of "splicing together" the observations made: each time an HTTP request is initiated by the browser, the plugin randomly generates a number called **httpid** that is inserted into the HTTP request header. Since the *httpid* value will be recorded both, by the plugin and packet capture, it can be used to correlate the information obtained from plugin and packet trace. The details of how to reconstruct a Web session from the packet trace are discussed in the next section.



(a) Plugin Records (b) Packet-Level Capture Records

Figure 2.2: Information Logged

2.3 Measurement

We first explain which performance measures we extract from a packet level trace and then report on three different experiments where we repeatedly access the same Web page (e.g. `www.youtube.com`) during one day.

2.3.1 Metrics and Methodology

A typical Web page normally contains tens, up to hundreds of objects that make up the whole page. To fully render the Web page, the browser needs to load all these objects; in the extreme case, this will require as many name resolutions and TCP connections as there are objects. In the case of YouTube, the whole Web page consists of about 30 objects stored on about 10 distinct servers. The typical procedure for loading one object is shown in Fig. 2.3: The time elapsed can be broken down into the following metrics:

- Name Resolution (DNS): The time elapsed between the first DNS query and first corresponding response with valid IP address(es) defines t_{DNS} .
- TCP Handshake RTT (SYNACK RTT): Since our measurement point is at the client side and all of TCP connections are initiated by the client, we define t_{TCPRTT} as the time between the first SYN packet sent by the client and its corresponding first SYN-ACK packet received from the server.
- Network RTT (Net.RTT): The time between the first data packet (normally carrying a GET request) sent by the client and its first corresponding ACK is referred to as t_{NetRTT}.
- Service Offset: When the ACK packet from the server in response to the data packet that carried the GET request does not contain any payload, there will be a time-gap between this ACK and the first data packet from the server, which is referred to as t_{Offset}. Since this gap between client request and data reception is caused a delay due to server processing, we call this gap as service offset.
- Data Transfer Period: The time between the reception of the first and last data packet containing the data of a Web object is referred to as $t_{DataTrans}$.

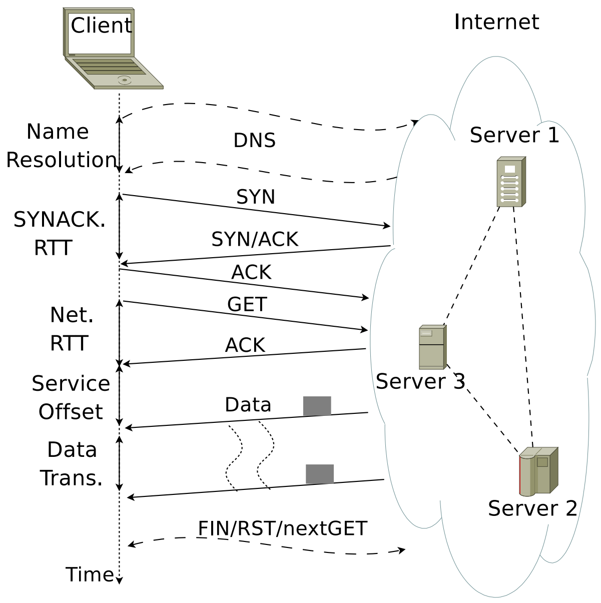


Figure 2.3: Metrics for Downloading One Web Object.

We measure two types of RTT, namely the TCP handshake RTT and the Net.RTT, which is useful in situations such as the following:

- In a whole Web session, one TCP connection can be re-used to fetch multiple objects, in which case we will not obtain a sample for t_{TCPRTT} for all of the object downloads.
- If the client is not directly communicating to the server because of an intermediate proxy, the SYNACK packet is usually come from the proxy and not the server and t_{TCPRTT} does not measure the round trip time between client and server.

When we analyze the the packet trace for one Web session we get a time series of values t^i_m where m denotes any of the metrics defined above (e.g. DNS, TCP Handshake RTT, NetRTT, etc.) and i refers to the *i-th* object loaded in that Web session. Given such a time series, we define the weight w_m of metric m as:

$$w_m = \frac{\sum t^i_m}{(\sum t^i_{DNS} + \sum t^i_{TCPRTT} + \sum t^i_{NetRTT} + \sum t^i_{Offset} + \sum t^i_{DataTrans}}$$

Based on the above weight definition, we use the full page load time T_{full} measured by the plugin to compute the **weighted breakdown** for metric m as:

$$breakdown_m = T_{full} \times w_m$$

where m is a metric such as DNS, Net.RTT, etc. From this breakdown definition, we can easily visualize the total "weighted" contribution of each metric. This breakdown definition is similar to the one in [63]. However, [63] uses different metrics and solely relies on the packet trace.

2.3.2 Measurement Result

In this section, we illustrate our approach via some experiments.

2.3.2.1 Wireless vs. Wired Client

The experiments are done in a private home on both, a wired and wireless PC sharing the same network access. We run the tests on two PCs simultaneously. In order to make the client fetch the Web content from the server, we automatically clear the browser cache after each browsing.

Load times of the YouTube sessions, where the main page of YouTube is requested, are shown in Fig. 2.4(a)-2.4(b). We see that most of the YouTube Web sessions in both, the wireless and wired cases, achieve load times of two seconds or less; However, in the wireless case, there are some outliers with very high values: The first impression time sometimes being larger than 5 seconds and the page full load time as large as 10 seconds. In Fig. 2.4(c)-2.4(d) (best viewed in color), we see that some of them have large handshake time and others may have large data transfer period. The explanation for these large values can be obtained looking at Fig. 2.4(e) and Fig. 2.4(f), which depict the Net.RTTs. For the wired case we see very stable values in the range between 20 – 100 ms. However, in the wireless case these values increase to 3 seconds. There are two possible explanations: (i) the wireless access point is overloaded or (ii) the quality of the wireless link is poor, which results in many link layer retransmissions.

2.3.2.2 Student Residence A

In the second experiment we have a client PC in a student residence. The client is connected via Ethernet and is sharing the access link from the residence to the Internet with a large number of other clients.

Fig. 2.5(a) depicts the page load times as experienced by the client. We see that the full load times are in the order of 5 seconds or higher. Some of the first impression times are also in the order of several seconds.

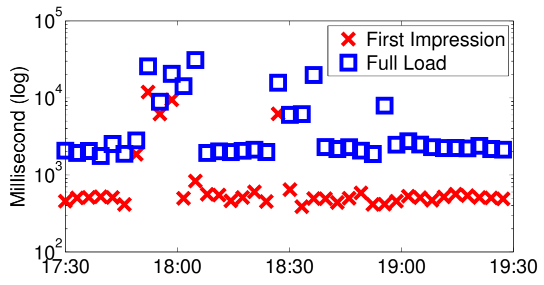
Fig. 2.5(b) shows the contributions of the different metrics that make up the load time of a Web page. What is striking are the high values for t_{DNS} and t_{TCPRTT} relative to the time $t_{DataTrans}$ it takes to download the data of the object. In Fig. 2.5(c) we plot the DNS response time and TCP handshake time values. We see that the values are either below 200 ms or above several seconds. Since the default timeout value for DNS and TCP SYN retransmission in the Linux client is 5 seconds and 3 seconds respectively, we can infer that the high values observed are due to packet loss and retransmission after timeout. Fig. 2.5(d) shows the retransmission rate during data downloading for all the sessions. We see that all sessions suffer from packet retransmissions, with a retransmission rate ranging from 1% up to 8%. Also, not shown here, most of the retransmissions are triggered by timeout and not by fast retransmit[1].

2.3.2.3 Student Residence B

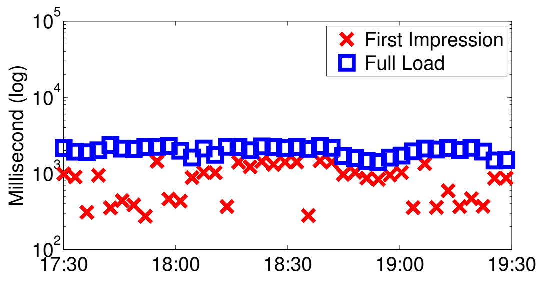
In the third experiment, we move to another student residence and the client PC is connected to Internet with its public wireless access point.

Fig. 2.6(a) depicts the client perceived results for the YouTube page. We see that, for both first impression and full load time are larger than 10 seconds in the first 2 hours (e.g. 22:00–00:00), and less

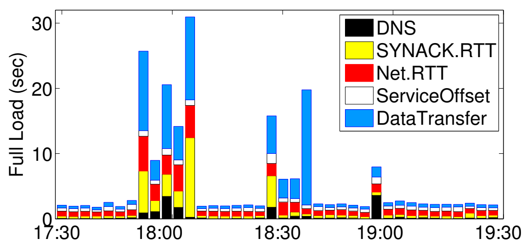
[1]We say *fast retransmit* as 3-duplicate ACK observed before retransmission; otherwise as *retransmission timeout*.



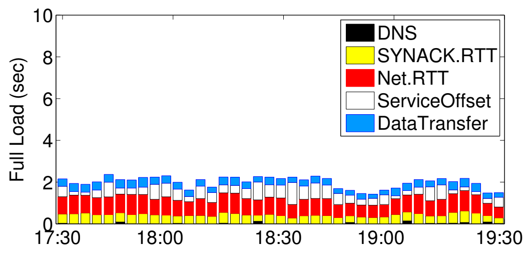
(a) User Perceived Results in Wireless Connection



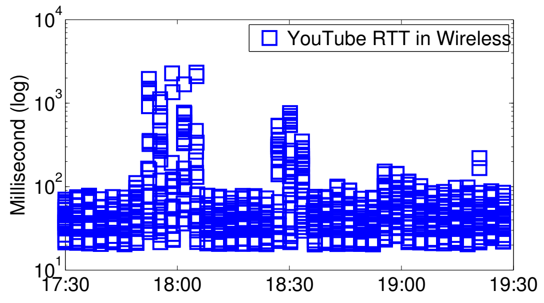
(b) User Perceived Results in Wired Connection



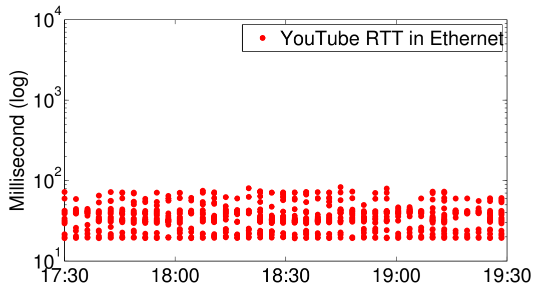
(c) Wireless Weighted Breakdown



(d) Wired Weighted Breakdown



(e) Wireless Net.RTT



(f) Wired Net.RTT

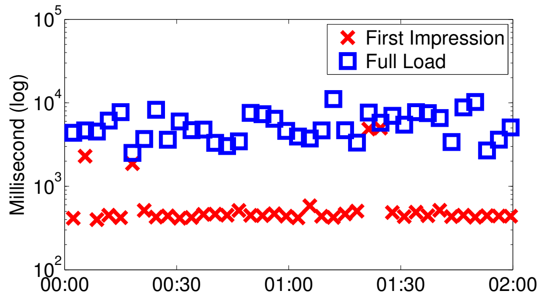
Figure 2.4: YouTube Sessions of Wired and Wireless Client.(starting around 2011-02-22 17:30)

than 3 seconds in the last few hours (e.g. after 2:00 am). We pick up parts of Web browsing with *good* and *bad* labels in Fig. 2.6(a) and show the weighted breakdown in Fig. 2.6(b)-2.6(c). From those breakdown figures, we see that:

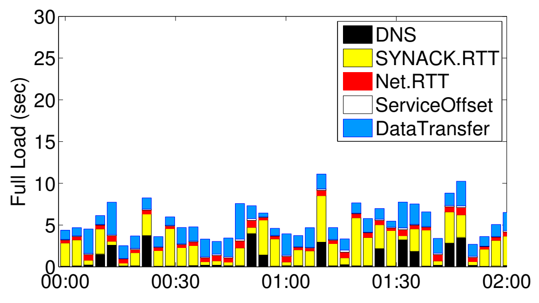
1. Service offset takes large portion in the *bad* sessions.

2. Handshake SYNACK RTT almost weights nothing for both *good* and *bad* sessions.

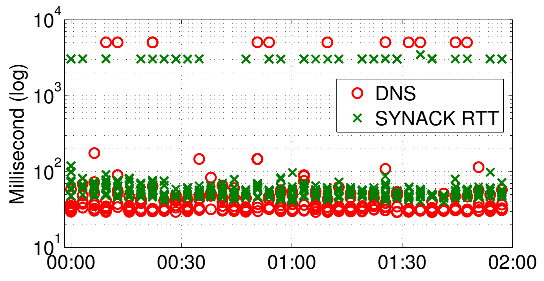
We start to investigate this behavior by the RTTs in Fig. 2.6(d). We see that SYNACK RTT is almost close to 0 ms, while Net.RTTs locate on two regions: one part that with the ACK payload=0 locates very close to 100 ms; while another with ACK payload size>0 are from 50 ms up to 100 ms. Based on this observation, we suspect that there should be a proxy close to the client and is used to establish TCP connections with client and such 100 ms should be a threshold of the proxy for each HTTP request: if the GET request does not receive corresponding data from remote Web server within 100 ms, as shown in Fig. 2.7(a), the proxy will send back the ACK to the client to avoid request retransmission; otherwise, as shown in Fig. 2.7(b), the proxy can directly send back the ACK together with the data.



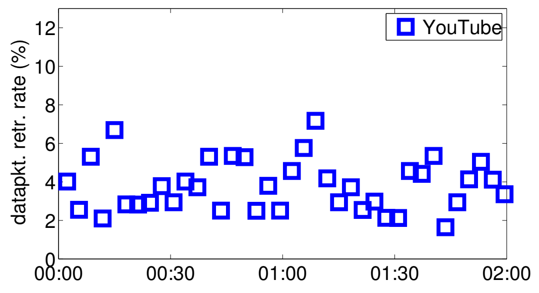
(a) User Perceived Results



(b) Weighted Breakdown



(c) DNS and TCP HS Time



(d) Packet Retransmission Rate

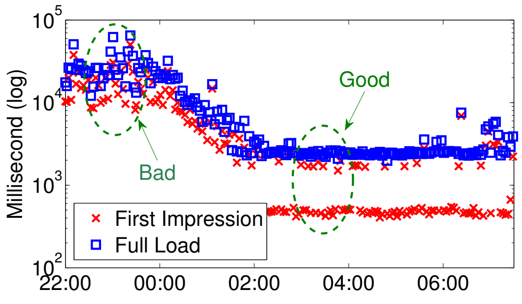
Figure 2.5: YouTube Sessions at Student Residence A. (starting around 2011-02-20 00:00)

Due to large weight for service offset in the whole page downloading, we report the results in time-series in Fig. 2.6(e). We see that large delays existing before 2:00 am while almost negligible afterward. Based on all the above discussions, we infer that bad performance for the Web browsing in this residence is due to the proxy side overload.

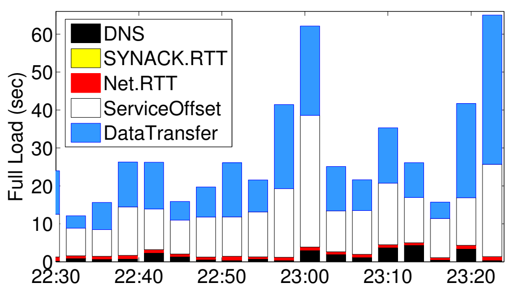
To learn more about this home network environment, we set up a Web server outside this residence and make the client PC inside the residence browse a particular Web page of our server. Meanwhile, we capture the traces at both client and server sides. We make such tests during both an evening period right after dinner and also early in the morning, we assume these periods as peak and non-peak Internet usage hours for the residence inhabitants. In Fig. 2.8, we show the measured handshake RTT (RTT_s in Fig. 2.7(a)) that captured at server side for both periods[2]. From Fig. 2.8, we clearly discover the differences, e.g. RTT values in both periods can have as large as 10 times differences in the median case. Such observation also illustrate possible overload behavior at the client side proxy.

Indeed, such proxy in this network environment can be considered as an HTTP proxy which is working on port 80. We show an example by collecting 1-hour random browsing trace from this residence, Fig. 2.9(a)-2.9(b) report different RTTs of this random trace for port 80 and 443. We see that connections for port 80 have similar RTT behaviors as discussed before, however, connections for port 443 have SYNACK.RTT and Net.RTT quite close with each other which indicates that no connection manipulation for that port.

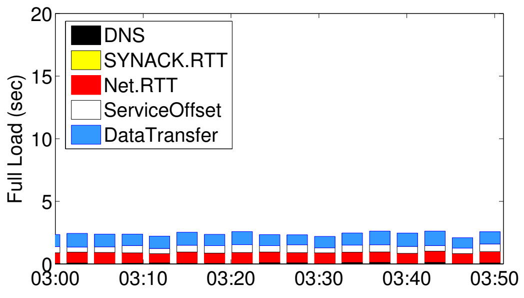
[2]We use RTT samples with no retransmission occurring during TCP handshake.



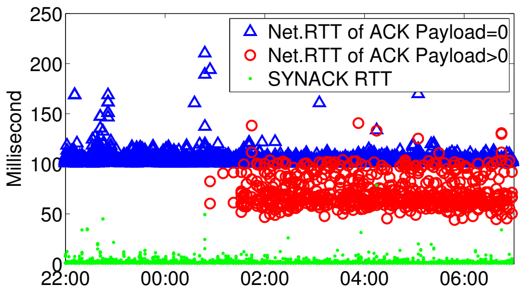
(a) User Perceived Results



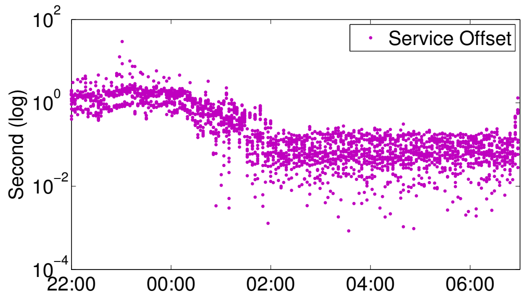
(b) Weighted Breakdown (*bad* sessions)



(c) Weighted Breakdown (*good* sessions)



(d) All RTTs



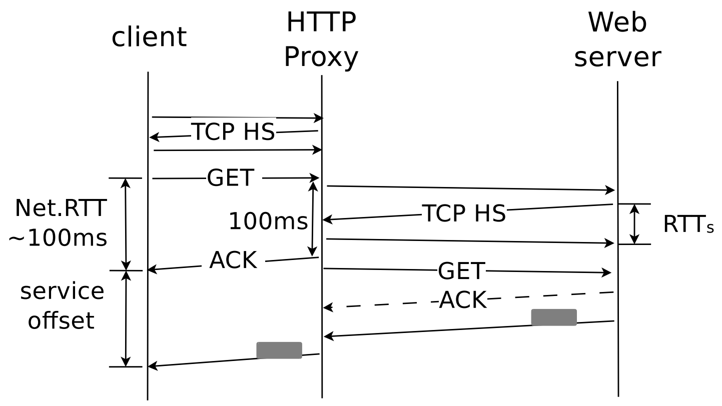
(e) Service Offset

Figure 2.6: YouTube Sessions at Residence B. (starting around 2010-11-23 22:00:00)

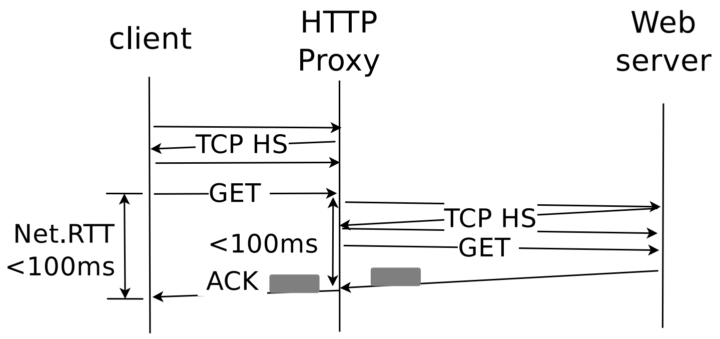
2.4 Related Work

Our approach is inspired by the work of Agarwal et al. [35] and Siekkinen [84]. Agarwal et al. developed the WebProfiler system that is composed of a browser plugin, WebProfiler service, a local and central database to diagnose problems in the Web service access. Siekkinen [84] developed the Intrabase system that uses database technology to efficiently post-process packet level traces. However, the aim of our work is different, namely we try to explain why Web access is slow, while the Intrabase system does a root cause analysis of long TCP connections. In our case the connections are typically short and we simply use Intrabase to analyze the loss or retransmission behavior of TCP connections.

Huang et al. [63] perform experiments in a controlled environment to study the performance of Web browsing on a smart-phone. There are also some related works that instrument the Web browser using a plugin such as Dynatrace [7] and Firebug [9]. These plugins can easily visualize the Web page structure and browser's behavior in rendering the page. The concepts of first impression and full load



(a) proxy does **not** receive data within 100 ms



(b) proxy receives data within 100 ms

Figure 2.7: Illustration of HTTP Proxy Case

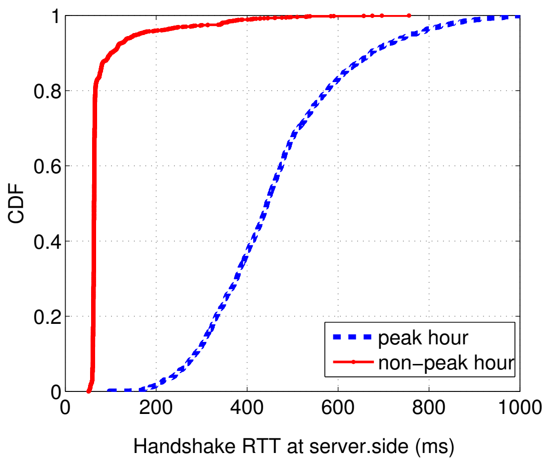
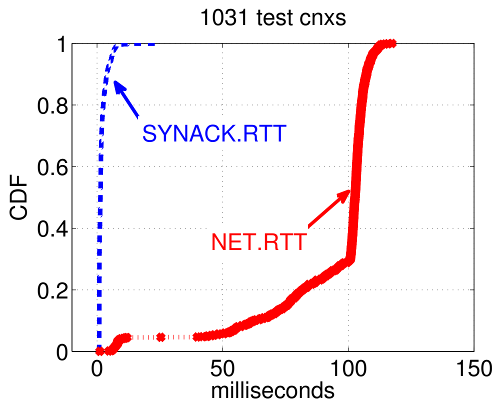


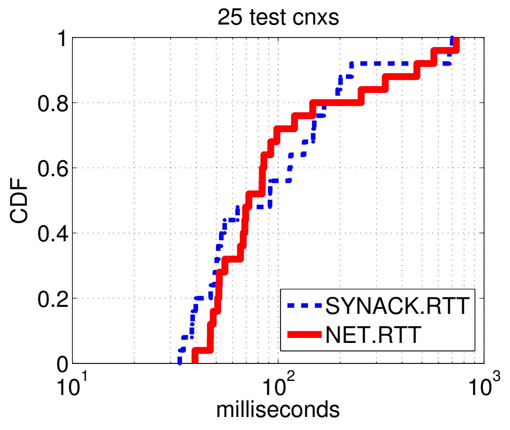
Figure 2.8: Server Side Measured Handshake RTT. (peak hour: 21:00 – 22:00, non-peak hour: 6:30 – 7:30)

are also proposed in dynatrace [7]. However, both systems do not perform any packet level analysis.

Plenty of systems are also proposed for home network management. Joumblatt et al. [67] propose HostView, which collects a set of end-host data such as network traffic, basic network configurations,



(a) TCP Connections of Port 80 (http)



(b) TCP Connections of Port 443 (https)

Figure 2.9: 1-Hour Web Browsing Trace at Residence B

and subjective user feedback to study the performance degradations. Calvert et al. [44] propose HNDR for *"general-purpose"* management and trouble shooting, and describe challenges and requirements for such system. Aggarwal et al. [37] propose NetPrints and Karagiannis et al. [69] HomeMaestro. Both systems are based on sharing knowledge between home users to solve problems. NetPrints focuses on the network configuration errors. HomeMaestro addresses performance issues caused by contention within home-networks, and combines observation with control in the sense of modifying the resource allocation to the different flows.

2.5 Discussion

In this chapter, we propose a method to combine observations captured at Web browser level with observations at packet level to detect and explain high Web access times. However, so far we have done only the first step and more work is needed to better understand the limitations and the potential for extensions of the approach.

We use two browser events to estimate the user satisfaction. We need to do some end user studies to confirm that real user's satisfaction can be measured this way. For this purpose, we have already implemented a user feedback bottom (c.f. Fig. 2.2(a)) that allows the user indicate whether he is satisfied or dissatisfied with the speed at which the current Web page was displayed. With the user feedback we also record the timestamp to be able to relate the feedback to the events preceding the feedback. We will illustrate our findings of this point in the next chapter.

As pointed by Yahoo![3], the scripting style in css or javascript can affect the browser rendering behavior and impact latency. In the experiments presented in this chapter, we sidestep this issue by downloading the same Web page over again.

[3]http://developer.yahoo.com/performance/rules.html

2.6 Conclusion

In this chapter, we have presented the initial architecture of a tool to analyze performance problems in Web access. The tool relies on measurements at browser and network levels to identify problems and explain their possible causes. The measurements of different end devices can be correlated to help reduce the number of possible causes and to improve the explicative power. Via some experiments, we have demonstrated the use of the tool and the insights it can provide.

CHAPTER 3
On the Relationship between QoS and QoE for Web Sessions

"*If your actions inspire others to dream more, learn more, do more and become more, you are a leader.*"

John Quincy Adams

3.1 Introduction

Traditional evaluations of the Web surfing mainly use Quality of Service (QoS) metrics that are easy to measure such as packet loss rate or round trip times. However, in recent years, the measurement community try to "move up the stack" and capture the real user experience, referred to as Quality of Experiences (QoE). In 2011, SIGCOMM even started to organize a special workshop discussing the problems in measuring user experience[1].

Based on the initial platform proposed in previous chapter, we evaluate user QoE during Web surfing and its relationship to QoS. In this chapter, we propose our methodology, such as clustering, to achieve the correlation between QoE and QoS; and also demonstrate that this clustering is suitable to compare and explain the experience among different clients and Web pages.

3.2 Measurement Set-up

3.2.1 Architecture

Based on the discussions in Chapter 2, we set up an experimental platform that uses a firefox plugin and normal packet capture to measure both, QoE related information such as user satisfaction and QoS related information such as full page load time, RTT, or packet loss rate etc.

[1]http://conferences.sigcomm.org/sigcomm/2011/workshops/W-MUST/

- The firefox plugin, among others, enables the user to express his **dis-satisfaction** with the time it took to render a Web page by clicking on an icon embedded in the browser. We also use the plugin to bind each HTTP query initiated by the browser to its associated firefox window/tab, which makes it easy to associate queries with the web sessions of the users and to measure the full load time for that Web page.
- We use packet capture (libpcap format) to obtain raw packet traces that are loaded into a database for post-processing.

3.2.2 Metrics

A typical Web page can contain up to hundreds of objects. To fully render the Web page, the browser needs to load all these objects. Similar as Chapter 2, we show typical procedure for loading one object in Fig. 3.1. As is shown in Fig. 3.1(a), for the download of each object, we extract from the packet trace the following metrics:

- **synack** is defined as time elapsed between the first client SYN packet and the corresponding ACK.
- **net.rtt** is defined as time elapsed between the first GET query and the corresponding ACK. These two metrics can be considered as the TCP connecting and HTTP query delays observed by the end client. We also measure the TCP handshake **rtt** where retransmissions of the TCP handshake packets will be excluded, and it can be considered as the round-trip delay between the client and Web servers [2].
- **service offset** is defined as the interval between the ACK of the GET request and the first data packet observed; in case that the ACK already carries a payload, service offset will be set to zero.
- Besides the related delay metrics, we also compute the data loss and retransmissions from the server to the client as is shown in Fig. 3.1(b). To estimate the retransmissions, we use method similar to [85], which is based on the TCP sequence number and the IPID field . The only difference is that, whenever we observe a retransmitted data packet, if such packet is not observed before, we consider it a **loss**; otherwise a **duplicate retransmission**.

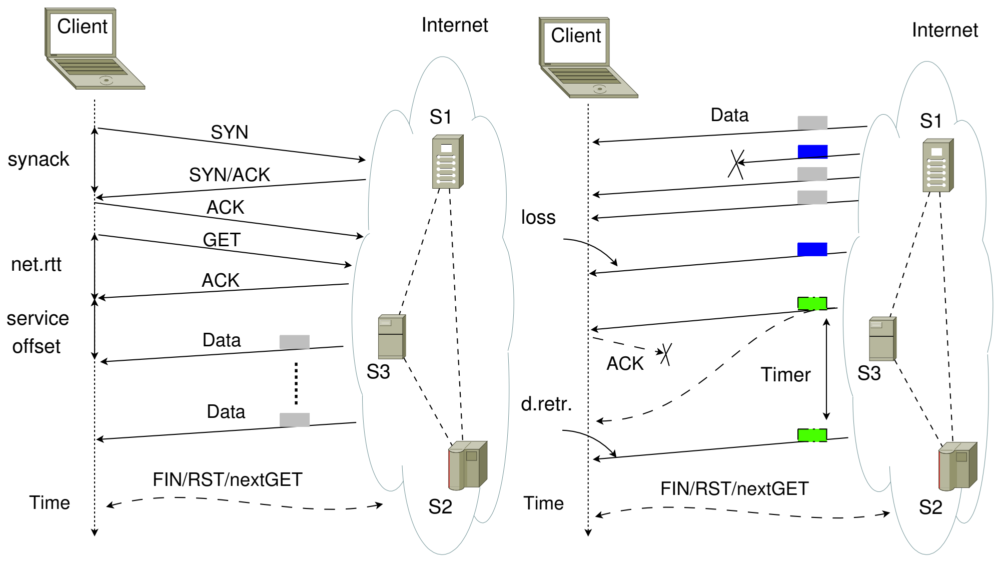
To describe the QoS of a Web *session*, we use the metrics computed for each Web *object* and compute the **mean** over all the values of the given metric to obtain a **Key Performance Index (KPI)**, which consists of

$$[nr., SYNACK, NET.RTT, SERV.OFF., RTT, LOSS, D.RETR.]$$

Note that *nr.* is the number of distinct GET requests during a complete Web session, which provides an estimation of the size of the Web page size in terms of the number of objects. We use upper case letter to denote means and lower case to denote individual sample values.

We label the average rate of such two types of retransmissions during a Web session as **LOSS** and **D.RETR** in the signature. LOSS can be used to estimate up-stream packet loss; since our measurement

[2]In case of a proxy terminating the connection requests of the client, the measured rtt will only refer as round-trip time between the client and the proxy, and not between the client and the server. However, in this chapter, we do not discuss the proxy case studies.



(a) Delay Related Metrics (b) Loss Related Metrics

Figure 3.1: Metrics Related to Download One Object In a Web Page

point is the client host, D.RETR can be caused by reverse ACK loss, or too short retransmission timers at the server side, etc.

Meanwhile, for the KPI, we focus on metrics captured by TCP connections and ignore DNS queries in this chapter. DNS pre-fetching is widely supported by recent browsers and this causes DNS queries to occur before a real web page surfing, which makes the lookup times less useful. However, we will discuss the effects of DNS (e.g. response time, response IP etc.) on the browsing experiences in following chapters.

3.2.3 Methodology

To process our results and reveal the underlying relationship between lower QoS-level and higher QoE-level metrics, we use clustering. Fig. 3.2 briefly shows the flow-chart to process our measured data. Since some of the values in the KPI are in different units and have different value ranges, we normalize all into the range [0,1]. For each type of metric *i* in the signature, we use

$$\frac{x_i - min_i}{max_i - min_i},$$

where x_i is the raw value for metric *i* in the signature; max_i and min_i are *maximum* and *minimum* values for that metric *i* respectively.

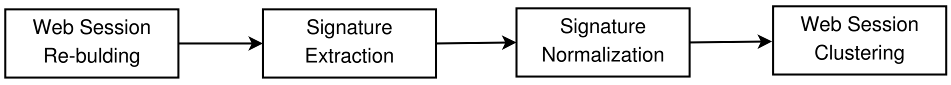
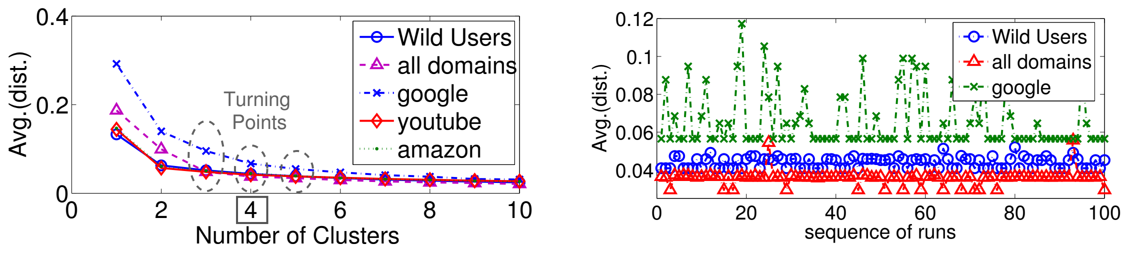


Figure 3.2: Data Process Chart

To cluster the results, we choose the well known **kmeans** algorithm, which is an un-supervised classification algorithm that does not need any training. The tricky point in *kmeans* is how to set a-priori the number of clusters. Hafsaoui et al. [59] use the dimension-reduced (t-SNE) method to determine cluster numbers, while other papers [52] [77] propose different algorithms for automatically detecting the right number of clusters. However, these methods are either based on pure visual inspection or introduce some extra parameters.

In our case, we carry out a comparison of the results obtained for different number of clusters. We define as **error distance** the squared euclidean distance between each sample in a cluster and its corresponding centroid. Fig. 3.3(a) shows the average error over all samples for different numbers of clusters based on the data sets we will use in the following. We see that for one or two clusters, the average error distances are relatively larger, and that the error distance rapidly decreases for three clusters or more. This observation indicates that a number of clusters which is between 3 and 5 is sufficient to achieve small clustering errors. Based on the above discussions, we mainly use *four* clusters and also discuss other cluster number issues in the chapter. Since the initial centroids are chosen randomly, in order to achieve local optimal results, we need to run the *kmeans* algorithm multiple times and choose the optimal one. Fig. 3.3(b) shows the distance error with 100 individual runs, and we see that the optimal results can be achieved within few runs. In this case, in the following work of this chapter, we just run 10 times *kmeans* and keep as result the one with smallest distance error.



(a) Cluster Numbers *v.s.* Distance Errors (100 independent runs for each cluster number setting) (b) kmeans Runs *v.s.* Distance Errors (4 clusters case)

Figure 3.3: A Priory knowledge Choices for Kmeans

3.3 Controlled Experiment

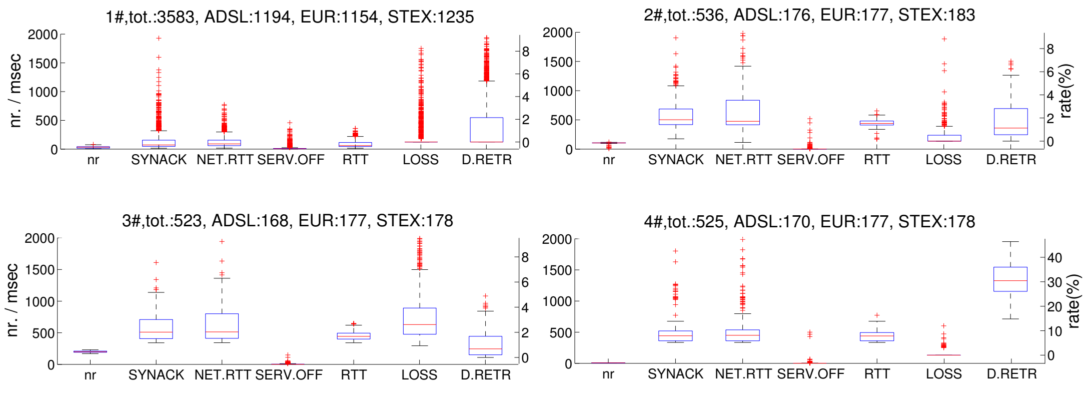
In this section, we predefine a list of Web pages, namely `www.google.fr(CDN)`, `www.amazon.fr(FR)`, `www.orange.fr(FR)`, `www.ebay.fr(FR)`, `www.youtube.com(CDN,US)`, `ccr.sigcomm.org/online(US)`, `www.sina.com(CN)`, `www.baidu.com(CN)`, `www.sohu.com(CN)`, `www.163.com(CN)`. These Web pages are diverse in geographical location and Web page type.

We emulate user browsing with three different clients that are (i) at home connected via an ADSL connection, (ii) in the office at Eurecom connected via a 100 Mb/s link to the Internet and (iii) in a student residence. All the machines are located in France. We name the ADSL home, the Eurecom office Ethernet and the student residence connections as **ADSL**, **EUR**, and **STEX** respectively. The experiments are done during the same evening in three homes and lasted for around 8 hours each. Both, the EUR and STEX client computers are connected via a wired connection, while the ADSL client computer is physically very close to the Access Point and uses a wireless connection. We clear the browser cache at the end of each Web session.

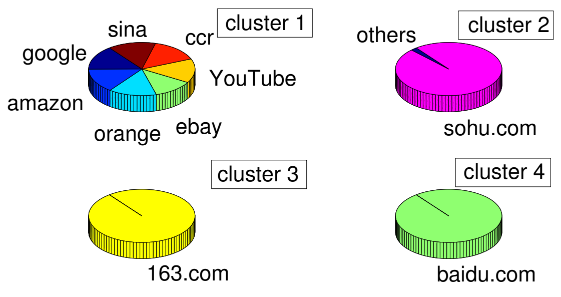
3.3.1 Global Analysis

For all the emulated Web sessions in three different locations we process and cluster the original data and present the results using boxplot[3], which provides the median and lower/upper quartile of a given metric. Fig. 3.4(a) shows the global clustering results of KPI values. Fig. 3.4(b) shows the Web page distribution as pie-charts and Fig. 3.4(c) shows the page load time of the different clusters. In Fig. 3.4(a) (also following figures), we indicate the cluster ID number by a number followed by a *sharp* sign in the title, and also indicate the number of sessions for a given location that are grouped in that cluster.

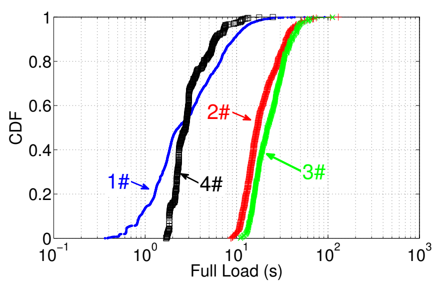
In Fig. 3.4(c) we see that the Web pages in clusters 1 and 4 have page load times that are typically in the order of a few seconds or less, while the Web pages in clusters 2 and 3 have page load times of 10 seconds and more. Clusters 2 and 3 contain almost exclusively requests to a single Web page (see Fig. 3.4(b)), namely `sohu.com` and `163.com`, which are both in China. The RTTs in these two clusters are normally around 500 ms and transfers in cluster 3 experience higher packet loss than in cluster 2; also the TCP handshake and HTTP query delays are large (see From Fig. 3.4(a)).



(a) Ctrl. Exp. Clustering Signatures



(b) Web Page Distributions in the Clusters



(c) Page Load Time

Figure 3.4: Clustering of All Ctrl. Sessions in 3 Homes

In cluster 4, the delay metrics for RTT, SYNACK and NET.RTT take values around 500 ms, while there is almost no loss; all the requests go again to a single Web page in China (`www.baidu.com`). However, data transfers in this cluster experience an un-usual large number of duplicated retransmis-

[3]http://www.mathworks.com/help/toolbox/stats/boxplot.html

sions, which are as high as 30%. Duplicated retransmissions can be caused by reverse ACK loss, congestion, or a too short retransmission timeout at the server. A more detailed investigation reveals that the time interval between two successive data packets with the same sequence number is around 200 ms, while the RTT is around 500 ms, i.e. the Web servers of `www.baidu.com` use too short retransmission timeouts, which results in a lot of spurious retransmissions. However, these premature timeouts do not seem to have a negative incidence on the full load time, which is four seconds or less for 80% of the Web sessions. Note also that the number of objects for this Web page is very small.

Cluster 1 contains all the other Web pages such as `google, amazon, ebay, orange, sina.com`. Web pages in this cluster normally have a small number of objects and their delay and loss metrics for this cluster are low. The whole page can be fully loaded within 10 seconds in 90% of the cases.

3.3.2 Discussion of Clustering Number

To compare the results with different number of clusters, we also plot cases with 3 and 5 clusters in Fig. 3.5. As we can see from these figures, the clustering still succeeds in isolating the Web pages with larger full load times (i.e. $>$10 seconds or larger) in a separate cluster.

3.4 Home Users Surfing in the Wild

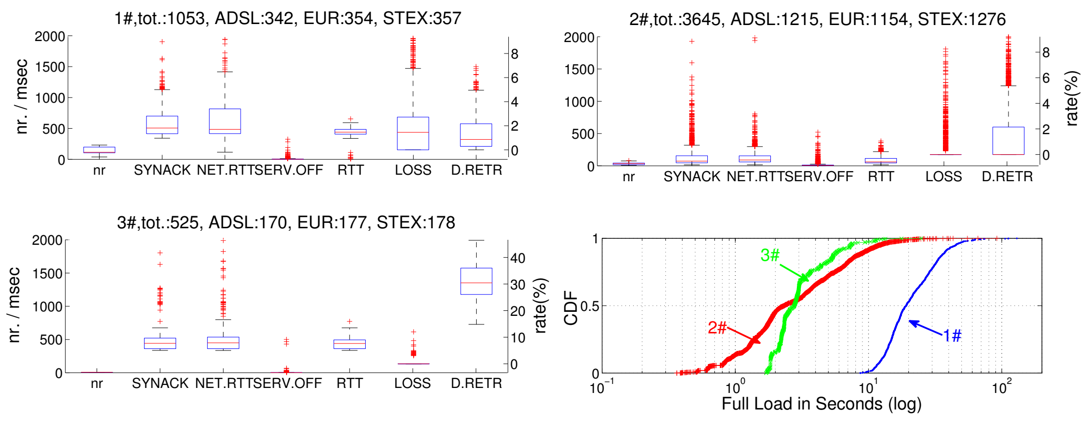
So far, we have pre-defined a number of Web pages that were accessed from different locations. We used clustering method to show the relationship between user perceived performance and lower level metrics. We now present a more "realistic" case, where clients are free to access whatever Web pages they want[4] and show how to apply clustering in this case to un-cover the underlying relationships between QoE and QoS related metrics.

Again, we pick three different locations, two of them are named ADSL and STEX which are the same clients from previous section, located in France. A third user, called TO is located in Torino, Italy. ADSL client uses a WLAN in his home, while STEX and TO users have wired connections. These users randomly browse web pages with diverse geographical locations and are asked to click on the **feedback button** of the browser plugin whenever they are dis-satisfied with the time it took to render the Web page. We make the users at least run the experiments for one night, then we manually collect all the data. We call a Web session **poor** when the users clicks on the dis-satisfied feedback button and **good** otherwise.

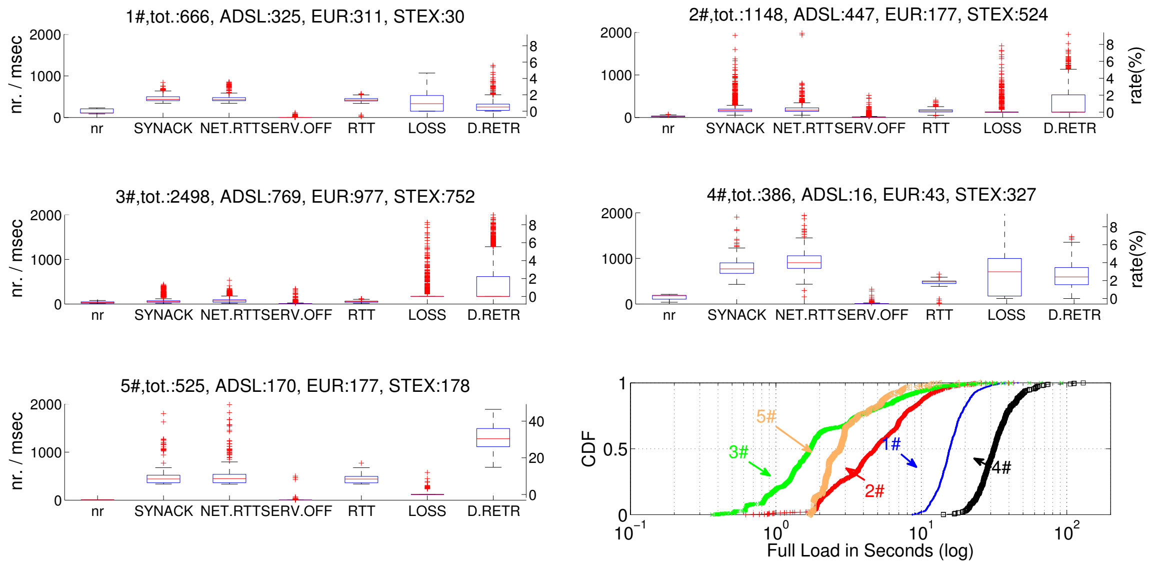
3.4.1 Global Analysis

We first take all the Web sessions of the wild users and cluster them globally. Fig. 3.6(a) shows the performance metrics for the different clusters. Since different users do not necessarily access the same Web pages, we provide geographical information to classify the Web pages. We use the country and organization database from maxmind [22], and for each cluster, we group all the requests and add the country name based on the IP address of the Web server. Since for Google, maxmind always returns US as country, we prefer to identify Google separately labeled as Google. Fig. 3.6(b) shows the geographical distribution via pie-charts and Fig. 3.6(c) shows the page load times for the different

[4]We focus on traditional Web pages and not personalized ones such as facebook, gmail, etc.



(a) All Ctrl. Sessions with **3** Clusters



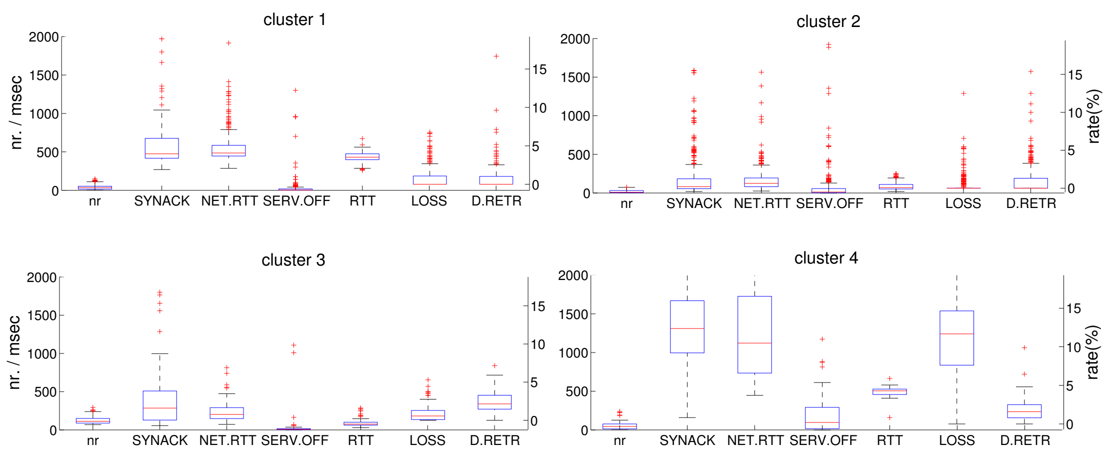
(b) All Ctrl. Sessions with **5** Clusters

Figure 3.5: ctrl.exp. for all domains with different cluster numbers

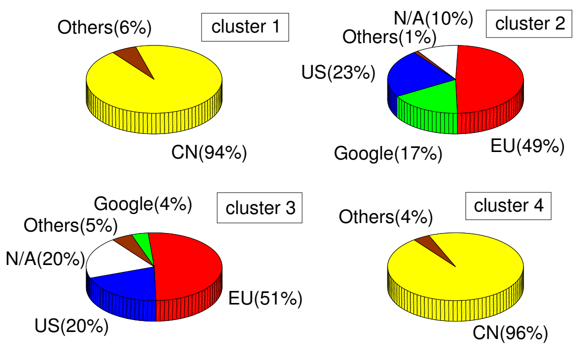
clusters[5]. Tab. 3.1 also lists median values for the different metrics and and the number of **poor** and **good** sessions.

- The Chinese Web pages are all concentrated in cluster 1 and 4, while the others are in clusters 2 and 3.
- The Web pages in cluster 2 have typically short full load times which is for more than 80% of these sessions bellow 10 seconds. The KPI metrics related to delay and loss are all quite low. In only 6% of the cases, users were dis-satisfied with the time it took to render the page, which is the lowest number of all clusters.

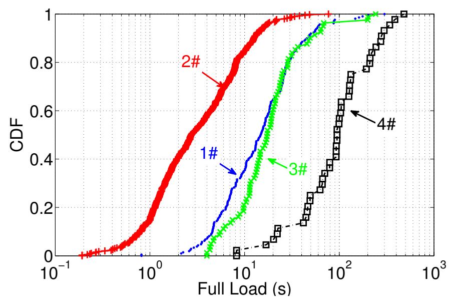
[5]Since real users may abandon a Web page browsing before it is fully loaded, in this case, we compute the time from start of browsing to the last bytes received from such Web page.



(a) **Wild** User Clustering Signatures



(b) GeoIP in Different Clusters



(c) Page Load Time in Clusters

Figure 3.6: Clustering of All **Wild** user Sessions in **4** Clusters

Table 3.1: Statistics of the **4** Clusters (*TOT* and *poor%* refer as total number of Web sessions and the percentage of poor sessions in that cluster, respectively).

	Median values								
clusterID	*nr.*	*SYNACK*	*NET.RTT*	*SERV.OFF*	*RTT*	*LOSS*	*D.RETR*	*TOT*	*poor%*
Cluster 1	34	474	485	3	**431**	0.0%	0.0%	209	21%
Cluster 2	10	82	124	12	70	0.0%	0.0%	399	**6%**
Cluster 3	**113**	285	202	8	**76**	0.6%	2.1%	81	42%
Cluster 4	44	**1310**	**1122**	96	**505**	**11.7%**	1.6%	44	**64%**

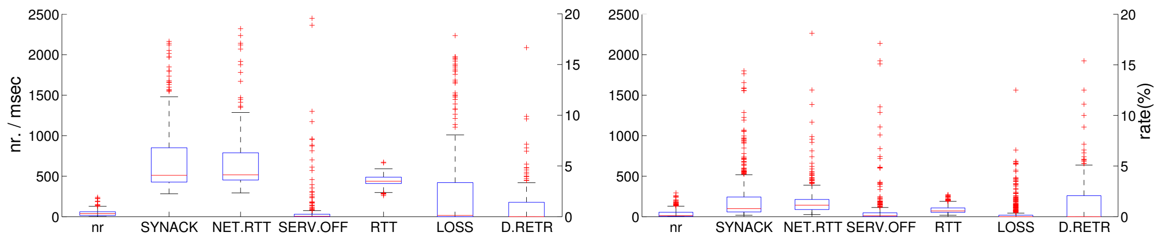
- On the other extreme, users were dis-satisfied in 64% of the cases of Web pages in cluster 4, which is not surprising, since 90% of these Web sessions need more than 10 seconds to fully load, and 50% of the sessions need even more than 100 seconds. The KPI metrics related to delay and loss are very high: The median delay for TCP handshake and HTTP query are over 1 second and the median loss rate is above 10%. After checking the details of the Web sessions in this cluster, we find most of them access to the same domain.

- Access to Web pages in Cluster 1 and 3 exhibits similar full page load times and are rated as poor 21% and 42% of the time, respectively. However, the characteristics of the Web pages and the KPIs for these two clusters are different: Web pages in cluster 3 normally have more objects than in cluster 1. From Tab. 3.1 we also see that the median RTT in cluster 3 is around 70 ms while for cluster 1 it is normally around 400 ms, which is not surprising since most of objects are from remote servers (e.g. CN, refer to Fig. 3.6(b)). The median of packet loss and duplicated retransmission rates in cluster 3 are a bit higher than in cluster 1.

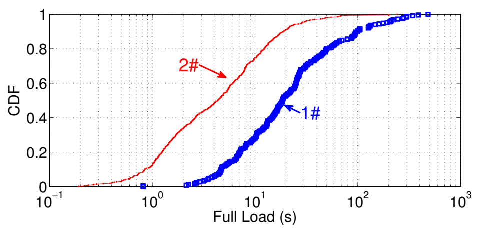
3.4.2 Discussion of Clustering Number

Similarly as before, the clustering number is also trivial for the underlying QoE and QoS relationships. Fig. 3.7 shows the clustering results for 2 clusters and Tab. 3.2 lists median statistic values and the annoyed sessions for different clusters:

- we can first discover a clear separation in the page load time for the two clusters in Fig. 3.7(b), around 70% sessions in cluster 1 need over 10 seconds to fully load the pages while 70% in cluster 2 take less than 10 seconds;
- median delay metrics such as RTT, SYNACK.RTT and NET.RTT in cluster 1 are normally as high as 400 ms while these delays in cluster 2 are normally around 100 ms;
- although loss in both clusters are around 0%, however, from Fig. 3.7(a), we see that loss in cluster 1 varies more than cluster 2 with around 4% as the upper quartile.
- combine all the Web page statistics, cluster 1 has 30% annoyed session while cluster 2 only has 11%.



(a) **2** Cluster Signatures



(b) **2** Cluster Page Load Time

Figure 3.7: **Wild** User Web Browsing in **2** Clusters

If we change the number of cluster to be 6, we get results shown in Fig. 3.8. Tab. 3.3 also lists the median statistics and users' annoyance ratio. From these results, we see that, by our clustering techniques, web sessions are also well separated for both their features and users' annoyance.

Table 3.2: Statistics of the **2** Clusters (*TOT* and *poor%* refer as total number of Web sessions and the percentage of poor sessions in that cluster, respectively).

	Median values								
$clusterID$	$nr.$	$SYNACK$	$NET.RTT$	$SERV.OFF$	RTT	$LOSS$	$D.RETR$	TOT	$poor\%$
Cluster 1	37	510	516	5	437	0.1%	0.0%	252	30%
Cluster 2	13	99	142	12	71	0.0%	0.0%	481	11%

- sessions with the largest page load time in cluster 4 have the most user annoyance ratio; Web sessions in that cluster normally have larger delays and higher loss than other clusters.
- cluster 1 has better delay behaviors for RTT, SYNACK.RTT and NET.RTT than cluster 4; loss in cluster 1 is also lower than cluster 4. However, since the Web sessions in cluster 1 normally have larger size and more objects are needed to render the page; therefore, Web pages in cluster 1 also take long time to be fully loaded, around 80% sessions have full page load time longer than 10 seconds.
- cluster 5 and 6 has similar users' annoyance ratio, while session features in these two clusters are quite different: cluster 5 has smaller page size than cluster 6; delay metrics in cluster 5 are larger than cluster 6; both loss and duplicate retransmission rates in cluster 6 are slightly higher than cluster 5.
- cluster 2 and 3 have the least users' annoyance ratio: more than 80% sessions in both clusters can be fully loaded within 10 seconds; median delays in RTT for both clusters are not so large as well; although some retransmissions can occur in both clusters, however, since page size are normally smaller than 10 objects, the final full page load can be still in the order of few seconds.

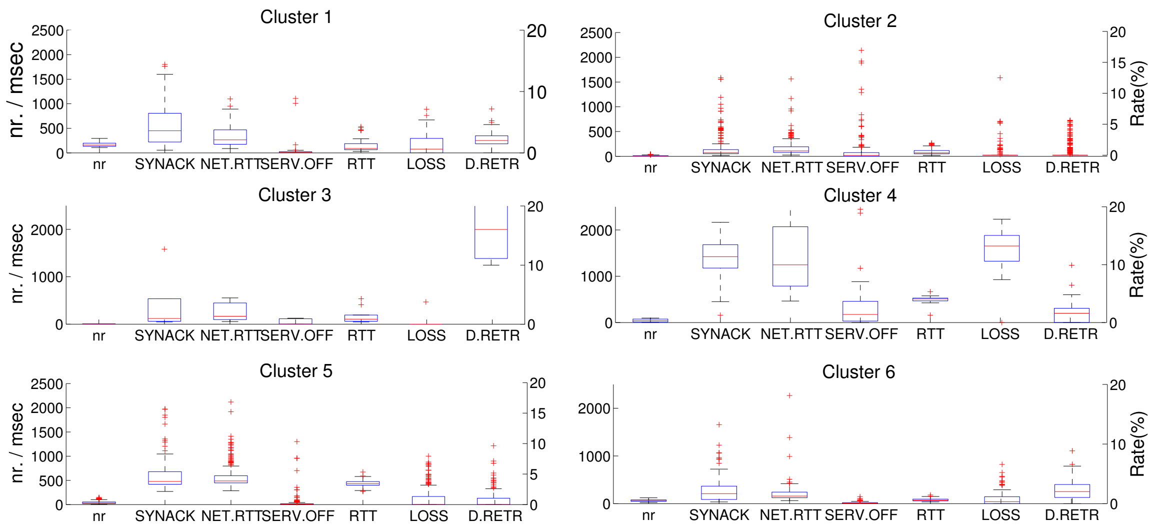
Table 3.3: Statistics of the **6** Clusters (*TOT* and *poor%* refer as total number of Web sessions and the percentage of poor sessions in that cluster, respectively).

	Median values								
$clusterID$	$nr.$	$SYNACK$	$NET.RTT$	$SERV.OFF$	RTT	$LOSS$	$D.RETR$	TOT	$poor\%$
Cluster 1	159	450	264	8	90	0.6%	2.0%	(24)/44	55%
Cluster 2	6	75	109	17	70	0.0%	0.0%	(10)/308	3%
Cluster 3	2	119	167	2	105	0.0%	16.0%	(1)/10	10%
Cluster 4	39	1422	1246	173	510	13.2%	1.6%	(22)/34	65%
Cluster 5	34	477	488	3	431	0.0%	0.0%	46/210	22%
Cluster 6	63	208	159	9	71	0.3%	2.0%	27/127	21%

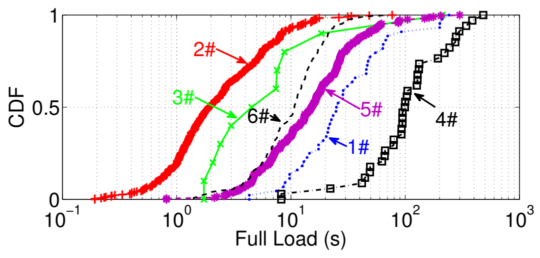
3.4.3 Per-Client Analysis

Since user dis-satisfaction is to a certain extent subjective, *we want to compare the KPIs of good and poor Web sessions on a per-user basis to see if we can identify commonalities among the different users*. The results are shown in Fig. 3.9:

- For the TO client, the performance differences between poor and good sessions are obvious: median SYNACK, NET.RTT, and RTTs for poor sessions are around 500 ms, while for good sessions, these values range from tens of milliseconds to 500 ms for the upper quartile. Loss and duplicated retransmission rates are slightly higher for poor sessions than good ones.



(a) **6** Cluster Signatures



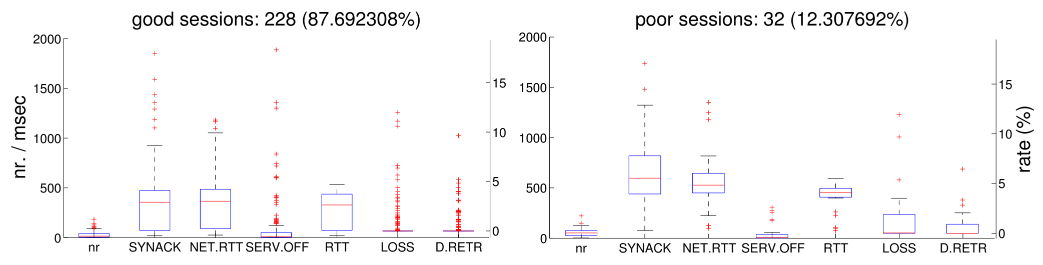
(b) **6** Cluster Page Load Time

Figure 3.8: **Wild** User Web Browsing in **6** Clusters

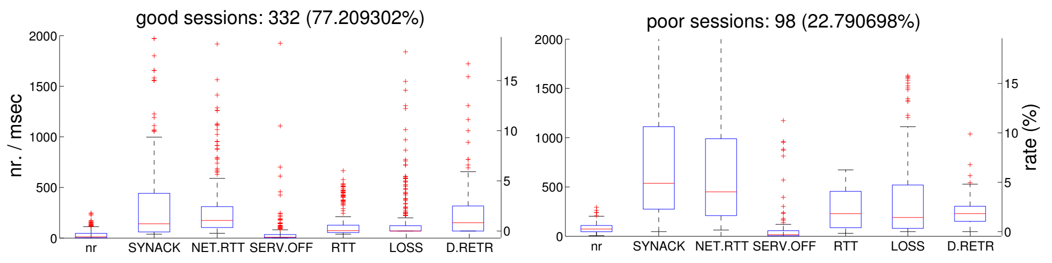
- For the STEX client, the situation is similar to the TO client: all the delay related metrics are much higher for the poor sessions than for the good ones, the same holds for the loss rates. If we compare the number of objects a Web page is composed of, we see that Web sessions rated as poor have a higher number of objects.

- The ADSL user never expressed any dis-satisfaction, which can be expected if we look at its KPIs, which are largely comparable and often even better than the ones for the good Web sessions of the STEX and TO client.

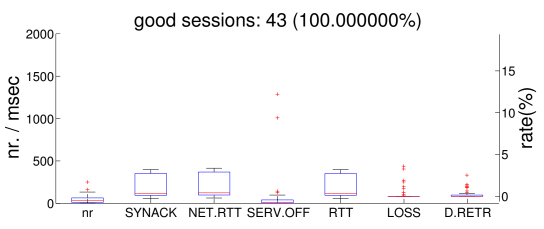
- Finally, if we look at the full load times (c.f. Fig. 3.9(d)) we see that the CDFs of the poor and the good session for both, the TO and the STEX client are very similar. Also around 90% of the Web sessions rated poor take over *10 seconds* to get fully loaded, while for all three clients between 70% and 80% of the session rated as good take less than 10 seconds to get fully loaded. These results are also similar with a conclusion made by an ITU-recommendation [65] that user's patient on their current Web page is around 10 seconds.



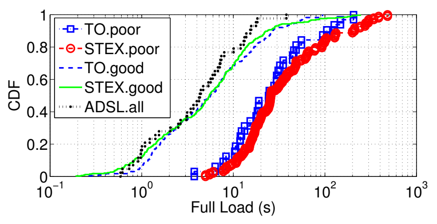
(a) **TO** User's Browsing Experiences



(b) **STEX** User's Browsing Experiences



(c) **ADSL** User's Browsing Experiences



(d) Full Load Time *v.s.* User's Complain

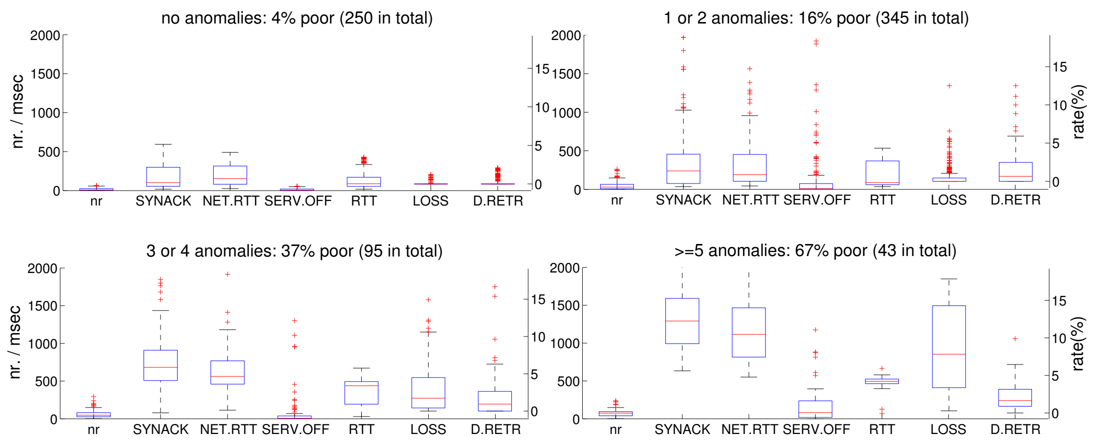
Figure 3.9: Performance Comparisons between **Good** and **Poor** Sessions for Different Users

3.4.4 Session Anomaly Analysis

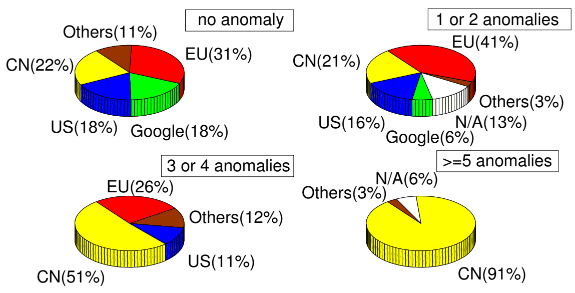
So far we used clustering to study the correlation between KPIs of a web session and the user subjective experience. We found a strong correlation between high values for some of the KPI metrics and user dis-satisfaction. As a complement, in this part, we focus on the **anomalies** in the collected wild user Web sessions: For each of the KPI metrics, we use the 80-th percentile as a threshold; if the value of a given metric is larger than its 80-th percentile, we consider that value as anomalous. For each Web session in our wild home user measurements, we count the number of anomalies in the KPI metrics and group the Web sessions by the number of anomalies' into: "no anomalies", "1 or 2 anomalies", "3 or 4 anomalies" and "more than 4 anomalies" to study the relationship between the number of anomalies and user dis-satisfaction. Fig. 3.10(a) presents the KPIs as boxplot, Fig. 3.10(b) gives the geographical distribution of the Web page objects in each group and and Fig. 3.10(c) shows the page load times.

We can see that Web sessions with three and more anomalies have full load times that are typically larger than 10 sec. resulting very frequently in user dis-satisfaction. These Web sessions are also characterized by high values for the KPIs related to delay and loss and most of these Web pages are hosted in China.

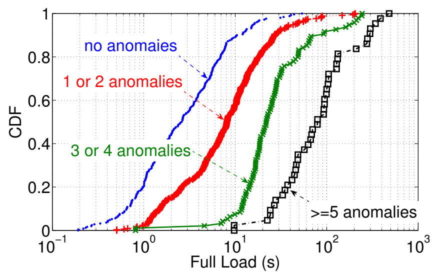
For Web sessions with no anomalies, we can see that the Web pages are normally small in terms of number of objects, also KPI metrics such as delays and loss are small. Page load times for these sessions are in nearly 90% of the cases below 10 seconds, resulting a very low percentage of user dis-satisfaction.



(a) Session Signature in Different Anomalous Groups



(b) GeoIP in Different Anomalous Groups



(c) Page Load Time

Figure 3.10: Session Anomaly Analysis for **Wild** Users

While the percentile-based method is complementary to the clustering approach, our goal remains the same, namely to group Web sessions according to their KPI values and to correlate user satisfaction with the values for the KPIs. Moreover, the clustering technique and percentile-based method are related. Tab. 3.4 shows the distribution of the number of anomalous KPIs for the different clusters presented previously in Fig. 3.6. We can clearly see that the Web sessions in cluster 2, which have the lowest percentage of dis-satisfaction, experience no or very few anomalies. A similar observation is true for the Web sessions in cluster 4, which have the largest percentage of poor Web sessions, and also the largest number of anomalies.

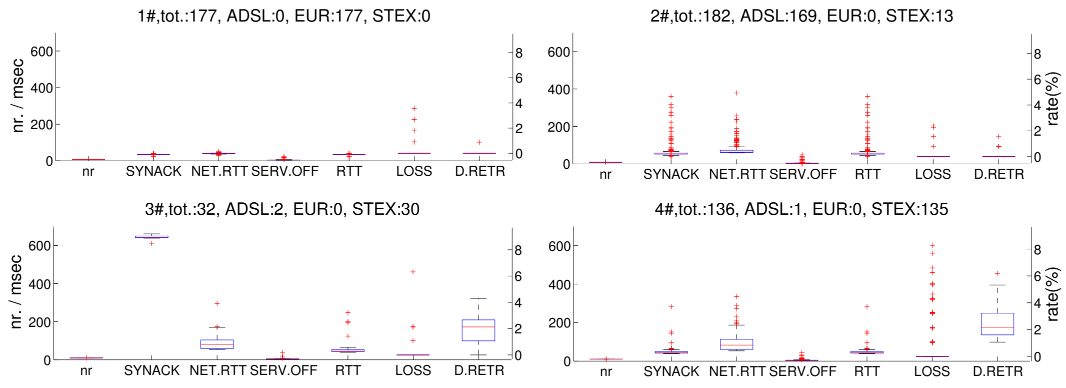
3.5 More Discussions

In this chapter, we use the platform proposed in Chapter 2 and a clustering method to quantify the relationships between QoE and QoS metrics. In fact, besides these studies, there are still some other

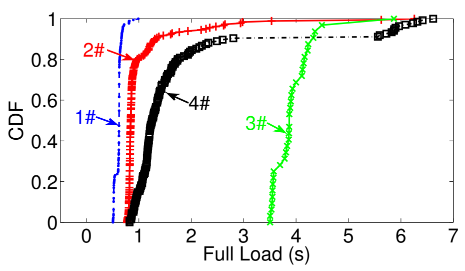
Table 3.4: Relation Between Clusters in Fig. 3.6 and the Number of Anomalies

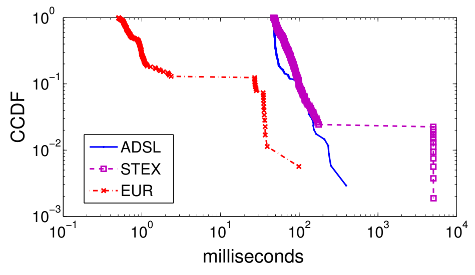
	Number of anomalies			
	0	1 *or* 2	3 *or* 4	≥ 5
cluster 1	22%	45%	28%	5%
cluster 2	51%	47%	2%	0%
cluster 3	0%	77%	21%	2%
cluster 4	0%	2%	30%	68%

insights that we can get from the clustering methodology. For example, since our experiment was done at three different locations, it allows to compare performance of the access to *the same Web page across different clients*. To illustrate this point, we use our previous controlled experiments discussed in Sec. 3.3 and select `Google` browsing data as an example, We show the results in Fig. 3.11.



(a) `Google` Clustering Signatures





(b) `Google` Page Load Time (c) `Google` DNS Response Time

Figure 3.11: All Controlled `Google` Sessions in 3 Homes

From the clustering results in Fig. 3.11(a), we see that, among the four clusters, requests from EUR are grouped in cluster 1 and from ADSL are grouped in cluster 2. On the other hand, the requests in clusters 3 and 4 are issued almost exclusively from the STEX client. The fact that the STEX client experiences higher delays and also loss in his local access can be seen in his KPIs. Cluster 3 is interesting because of its large SYNACK values; it turns out that for cluster 3, on average, one out of five SYN requests to establish a TCP connection does not get answered and must be retransmitted. Since the retransmission timeout is 3 seconds, we get SYNACK = $\frac{(50+50+50+50+3050)}{5} = 650(ms)$.

The long tail for the page load times in cluster 4 is due to the long DNS response time for some of the sessions. As we have already discussed, the STEX client experiences more packet loss than the other two clients. We can clearly identify in Fig.3.11(c) that around 3% of the DNS queries need to be retransmitted for the STEX client.

3.6 Related Work

This work builds on our initial platform discussed in Chapter 2, where we presented the measurement architecture but did not carry out any systematic analysis of the measurements.

Different methodologies of collecting user feedback are proposed in the literature. Chen et al. [45] propose the OneClick platform, which carefully evaluates the correlation between the user click rate and some controlled network performance metrics such as packet loss etc. Joumblatt et al. [67] propose HostView, which collects user feedback via an "annoyed" clicking icon and a feedback form; a follow up paper [68] discusses lessons learned and some preliminary results collected via HostView. SoylentLogger [74] is another system which is similar with HostView in measuring people and network data.

Compared to these works, our work focuses only on Web browsing, the properties on the Web pages, and the relation between KPIs and user satisfaction. Our analysis methodology is also inspired by work of A. Hafsaoui [58], where similar method is used to analyze the performance of TCP connections. Our work can be considered exactly as one of the *future work* proposed by Hafsaoui *leaving the connection level of analysis* and moving up to real user's level.

3.7 Conclusion

In this chapter, we have studied the case of Web browsing for the relationship between low QoS level such as delay and loss and high QoE level such as user satisfaction. We use both, controlled and wild browsing experiments and show that (i) user perceived performance and also subjective satisfaction can be somehow explained by lower level KPI metrics. By using machine learning techniques such as clustering, we can somehow un-cover such relation; (ii) when focusing on poor Web sessions only, clustering or percentile-based methods allow to explain user dis-satisfaction by looking at the KPIs of the different clusters; (iii) by comparing the values of the KPIs for good and poor Web sessions, we can identify the reasons for dis-satisfaction; (vi) we also see the relationship between the full load time and user's satisfaction that: a full load time of *ten* seconds or more is typically not acceptable for our users.

CHAPTER 4
Distributed Analysis and Diagnosis of Web Performance

"The only thing that will redeem mankind is cooperation."

Bertrand Russell

4.1 Introduction

From the discussions in Chapter 3, we see that one of the promising perspectives for clustering methods is to cooperatively diagnose performances across different homes (c.f. Sec. 3.5). Therefore, in this chapter, we show how to extend the clustering work to a distributed fashion. Our proposal can cope with some of the limitations of the previous chapter, for example, by avoiding the exchange of raw performance metrics, no centralized repository for the data, etc. These features make the clustering flexible for larger scale operation; but on the other hand, it somehow hides the user's privacy in terms of their raw performance values.

Our contributions include: (i) the novel idea and algorithm for cooperative analysis for Web browsing performance; (ii) a simple architecture that easily measures the key performance indices; (iii) we deploy our system in several real homes and evaluate it on a list of popular Web pages. Our results demonstrate that, even with limited knowledge, we can still find interesting insights to explain the Web performance across different homes.

4.2 System Design

This system for cooperation across multiple users is based on a distributed clustering algorithm which was proposed in [49]. In this section, we will briefly describe the key mechanisms of the algorithm, our selected features, and also the system architecture.

4.2.1 High-Level Algorithm Description

The main idea of the distributed clustering algorithm is to exchange **not** the raw information of each user, but only an "abstract" of the information. The main process for the algorithm works in the form of Gossip [16] as follows:

1. The clustering is always initialized by a user/peer, who then acts as a master. This master peer first computes a set of initial centroids V_1 based on its local data, and selects a positive threshold, γ, to be used for the clustering termination checking. After that, the master peer will send out the information together with the iteration number as a *poll* message to its neighbors. From this point on, the clustering activity begins.
2. After receiving a *poll* message for a new iteration at any peer of the system, if the current peer is not in its termination state, it will run one round of the traditional kmeans on its local data to produce its local centroids and cluster counts, then send these to all its neighbors.
3. When each peer has received all its neighbors' local centroids and their cluster counts, it will update its global centroid $V_{i,j}$ of cluster i, iteration j, by a weighted average computation.
4. At any individual peer, whenever new centroids have negligible variance from the previous iteration (e.g., $max\{||V_{i,j} - V_{i,j-1}||\} < \gamma$), that peer will terminate its own clustering operation and its local computations will not be updated any more. The whole clustering finishes when all the peers are in the "termination" status.

Here, we just briefly list the main idea behind the distributed clustering algorithm (for the formal definitions and operations, the reader may refer to [49]).

As initial parameters, the master peer that initiates the distributed clustering should choose the number of clusters and the threshold value γ for termination checking. As we have discussed in Chapter 3, the cluster number is not a big deal for separating differences in performance, and it only depends on the granularity that we need. In this case, in the following work, we still use **four** as our cluster number. For the termination threshold, we will illustrate with more details in the following.

4.2.2 Architecture

As shown in Fig. 4.1, the cooperation system is generally composed of two parts: the individual client peer and a public system coordinator.

4.2.2.1 Client Peers

Each individual client is instrumented with a Firefox browser plugin and the clustering module. The plugin performs a browser-level measurement and stores the performance metrics into a local database; the clustering module is implemented with the distributed clustering algorithm for diagnosis and analysis.

Measurement and storage Unlike Chapter 3, the measurement engine in each client only relies on a browser plugin. Some more functionalities by the packet capture discussed in Chapter 3 are integrated into the plugin. We use Mozilla development API [23] to monitor not only the page load performance, but also individual object download activity (described later in Sec. 4.2.3). Moreover, it periodically dumps the captured data into a local database. Here, we use SQLite as the database due to its light-weight relative to the storage file size and its fast access [6]. Finally, the plugin provides more interfaces to allow Firefox browser parsing arguments in the browser and control the browsing activity.

Clustering Apart from the measurement engine, another key component is the clustering module, which performs distributed clustering for peer collaborations. By default, the clustering module passively listens for a request. Whenever it receives a clustering request by an initiator, it will become active and start to fetch the corresponding metrics from the local storage according to the request parameters, and then, perform the local computations for the clustering process. Currently, a clustering request by the initiator contains information such as:

- Web domain key word string;
- starting and ending time for the corresponding data that are used for the clustering;
- number of clusters and the termination threshold value for the clustering algorithm.

For all the involved clients in the system, any one can initialize clustering by registering at the system coordinator (described in the following). If the clustering process has started, the distributed client peers communicate with each other through the system coordinator.

4.2.2.2 Public Coordinator

The system coordinator gives access to every client who wants to be part of clustering. It keeps an *active peer list* to maintain system communications. A *test* probe is sent periodically (e.g., every five seconds) to all clients to detect peer aliveness. In case there is no response from a client, a "peer-lost" message will be broadcast to others and make them update their neighbor information. In addition, the system coordinator also works as a relay server for packet forwarding to allow communication between clients even they are behind the NAT (Network Address Translation). Details of the controlled message and formats can be found in [46] [41].

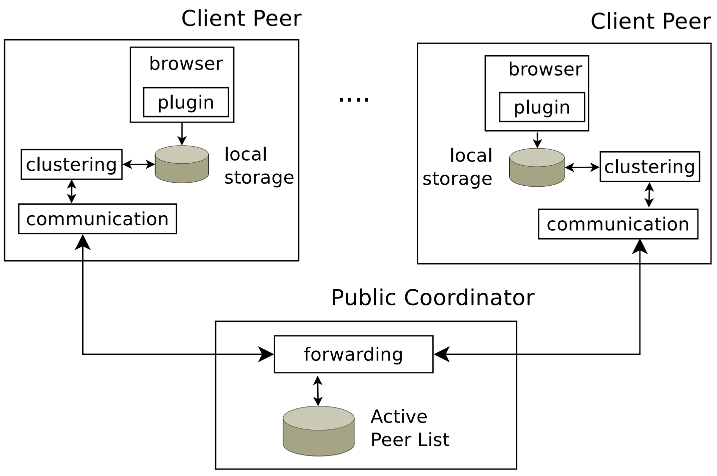


Figure 4.1: System Architecture for Dynamic Distributed Clustering

4.2.3 Selected Features

As was discussed previously, our measurement is achieved within the browser plugin. To illustrate the key metrics for our analysis and diagnosis, we show this in Fig. 4.2: when a given Web page is accessed, the browser will start to fetch the objects that make up that Web page. In this case, there will be different status events appearing in the browser.

- In Fig. 4.2, the downloading activity is started at t_1 with a DNS name resolution. The time elapsed between the DNS query (t_1) and its response (t_2) will be referred to as the **DNS delay** (dns=$t_{1,2}$). During such a period, a "`looking up`" text message appears in the browser's status bar.
- After the DNS lookup, at t_3, which corresponds to sending a SYN packet, a "`connecting to`" text message appears in the status bar until the client receives the SYN/ACK packet (t_4), We refer to this time interval as the **TCP connecting (or handshake) delay** (tcp=$t_{3,4}$).
- Whenever the browser detects that its TCP connection is established, it will immediately change its status (t_6), "`waiting for`" appears in the status bar, and an HTTP query ($t_6^{'}$) is sent. While we are waiting for the HTTP response data from the web server: The Web server can either directly send back the data ($t_8^{'}$), or first send back the TCP ACK ($t_7^{'}$) and then return the data. However, at the browser level, we can only capture a browser status event that will be triggered at t_8 when receiving the first data. We refer to $t_{6,8}$ as the total **HTTP query delays** ($http$=$t_{6,8}$).
- After a successful HTTP response, the browser keeps downloading the object data from Web servers until it is finished at t_9. The time period between reception of the first (t_8) and the last data packet (t_9) is the data receiving delay, and the browser status bar during this phase displays the text message, "`transferring data from`".

  Since this data receiving delay can be affected by many factors, such as network congestion, slow server response, etc., this is more obscure than the previous ones as far as diagnosis is concerned. In this case, we use the previous three metrics for our clustering process.

For a whole Web page downloading, since multiple objects and connections are involved, we take the average value of these selected metrics:

$$[\, DNS, TCP, HTTP \,]$$

Unlike centralized clustering, the distributed method, in practice, can not be run multiple times to choose the optimal results. In this case, we overcome the outlier sample effect (e.g., long-tailed distribution) by smoothing them using the logarithmic values of the continuous variables. To avoid special cases where the raw average metric value is zero, which would make the logarithmic computation nonsense, we take

$$[\, log(DNS+1), log(TCP+1), log(HTTP+1) \,]$$

as the input for the clustering process.

4.3 Real World Deployments

To evaluate our system and show its usefulness, we deployed it in four different homes. Three of them are located in the French Riviera and the other one is in Naples, Italy. Tab. 4.1(a) summarizes the

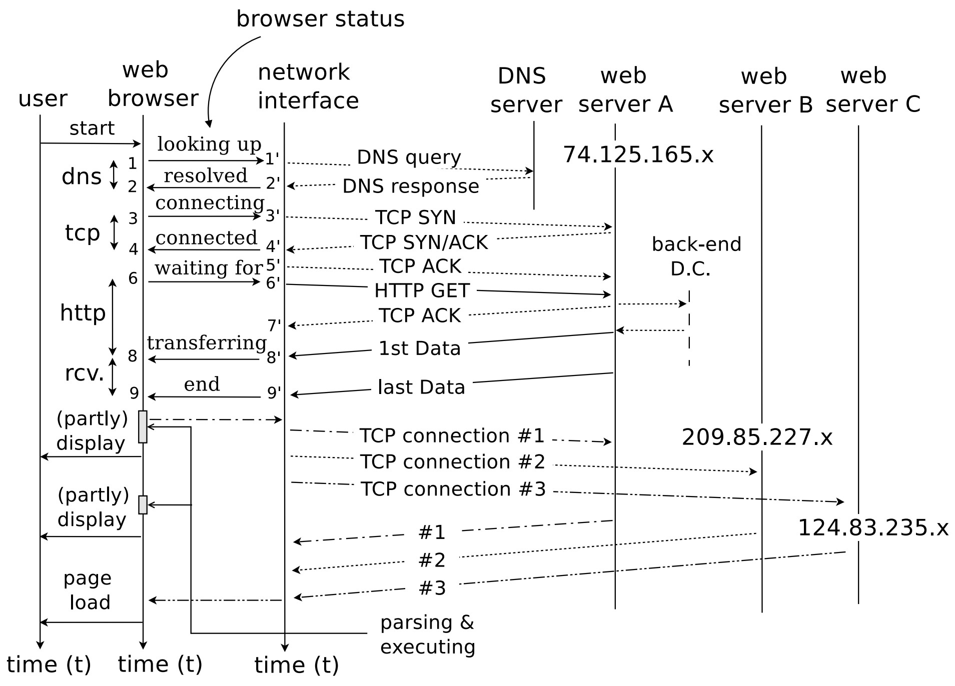


Figure 4.2: Measured Metrics by Browser Plugin

basic information for these homes and we see that all this set of locations is a diverse one regarding the properties of the homes and their network conditions. Moreover, all client PCs were instrumented with Linux OS and the same version of Firefox, 12.0.

For browsing activity, we defined a list of Web page URLs as shown in Tab. 4.1(b). We see that these are normally **popular** Web pages (or domains) for different countries ranked by Alexa[1]. and their properties in terms of size or number of servers are also diverse. To emulate the user behavior of Web browsing, we automated the Firefox plugin to control the browsing process. Fig. 4.3 shows the pseudo-code for our Web browsing emulation. Since users normally can use `google` or `bing` pages for their searching purposes, therefore, each time we browse these two pages, we randomly generated a five-character key word and added it into the URL, then browsed it. Each client PC independently executed the browsing process until we terminated it. For our next results, we kept browsing for around 16 hours in each locations. In case of anomalies that prevent a Firefox *full load* event from firing, we used three minutes as the timeout for each browsing. Before starting our experiment, we also disabled Firefox IPv6 DNS query, set the `Home Page` to blank, while other settings were set to default. Although this emulation may not necessarily model typical user browsing, we believe it is sufficient to evaluate our prototype.

4.3.1 Evaluations

We ran our browsing experiments for around 16 hours, from 8 PM Dec. 7, 2012, to midnight of the following day, and all the clients ran the experiments independently. In this section, we analyze the experimental results to evaluate the system and the next section will provide case studies on the

[1]www.alexa.com traffic rank in country.

Table 4.1: Basic Information for the Distributed System Deployments

(a) Home Information

Measurement Locations	*Connection Type*	*CPU/MEM*
EURECOM (EUR)	Ethernet wired	3.2GHz/8GB
Private Home (HOME1)	ADSL wireless	2.8GHz/4GB
Student residence (HOME2)	Shared WIFI	1.7GHz/1GB
Univ. of Naples Federico II (NAP)	Ethernet wired	2.0GHz/8GB

(b) Tested Web Page Information

WebPage	*Country*	*Rank*	*#Obj.*	*Vol.(KB)*	*#Servers*
www.google.fr/search?q=	US(Global)	1	9	283	4
www.bing.com/search?q=	US(Global)	13	2	19	1
en.wikipedia.org	US	7	36	297	4
www.lemonde.fr	FR	16	65	592	14
www.orange.fr	FR	10	32	250	7
www.virgilio.it	IT	14	116	1,102	24
www.libero.it	IT	6	61	509	18
news.baidu.com	CN	1	99	1,132	21
tech.sina.com.cn	CN	4	132	1,873	28
www.zaobao.com.sg/edu/edu.shtml	CN(SG)	253	72	334	13

```
1: loop
2:     $\$URL \leftarrow$ Pop up one URL from the web list
3:     if $\$URL \subset$ google or bing pages then
4:         $\$RandStr \leftarrow random\ string$
5:         $\$URL \leftarrow \$URL\ +\ search?q = \$RandStr$
6:     Launch Firefox to browse $\$URL$
7:     while $fully\ load$ or $timeout$ do
8:         Quit Firefox & Clear cache
9:     sleep for 5 seconds
```

Figure 4.3: Pseudo-Code for Browsing Process

performance analysis and diagnosis.

4.3.1.1 Choice of Threshold

As discussed in Sec. 4.2.1, we need a pre-defined threshold for the maximum centroid movement between two consecutive iterations to determine the termination of the clustering process. It is obvious that the smaller this threshold value is, the more iterations we need before termination, while the samples converge better to the final clustering centroids. In order to choose a better trade-off value, we investigate the relationship between the threshold value and the clustering iterations. We randomly chose several Web pages and did the distributed clustering with multiple threshold values. For each clustering process, we did it 10 times and computed the average number of iteration rounds for each threshold setting. Fig. 4.4(a) shows the results of clustering on the first four hours of data, and Fig. 4.4(b) shows the results for the entire experimental period. From these results, we can see that:

- The clustering iteration does not depend on the sample size. We see that there are no obvious differences between 4-hour and 16-hour browsing data, although the longer time spent much more browsing samples.
- The clustering iteration rounds depend on the threshold value. When it is smaller than a certain value (e.g., < 0.1), the rounds needed keep increasing; meanwhile, it does not change much when it is larger than 0.1.
- In general, the number of iterations for the distributed clustering is small: the maximum iteration for all these tests is only around 10. This also indicates that, compared to centralized clustering which needs all the raw sample values from all the other peers, the distributed method needs much lower overhead if the other peers have large data samples.

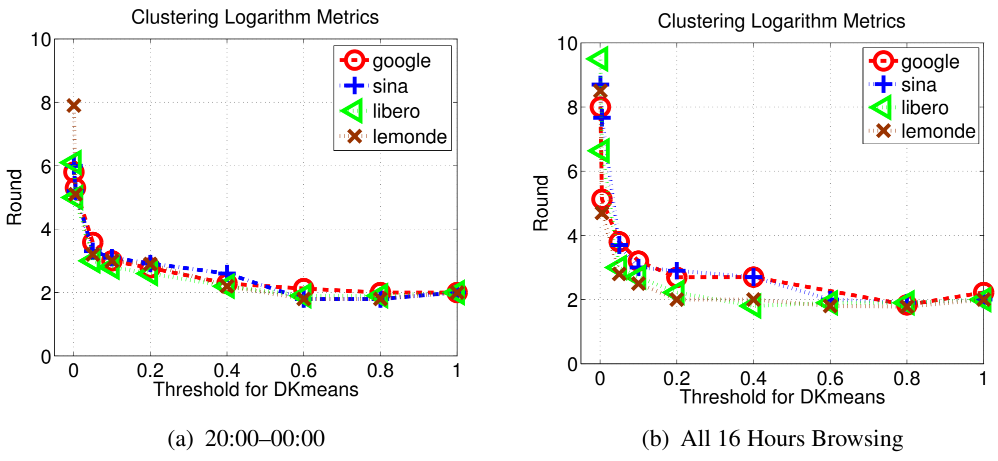


Figure 4.4: Threshold Discussion for Real Home Web Browsing (tested threshold values: 1.0, 0.8, 0.6, 0.4, 0.2, 0.1, 0.05, 0.005, 0.001, HOME2 client is the master peer and the iteration rounds are computed based on the master peer.)

Based on the above results and discussion, in the following parts we use **0.1** as the threshold to determine the clustering termination.

4.3.1.2 Clustering Quality

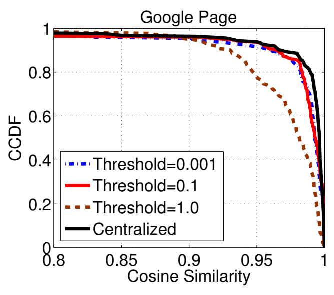
After determining the threshold for the distributed clustering, we now evaluate the clustering results and also compare them with the centralized method. To quantify the quality of the clustering results, we use the term **cosine similarity** [4]: given two samples that can be represented by the vectors $A = \{a_1, a_2 \cdots a_n\}$ and $B = \{b_1, b_2 \cdots b_n\}$, the similarity between them is

$$Similarity = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} a_i \times b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2} \times \sqrt{\sum_{i=1}^{n} (b_i)^2}}.$$

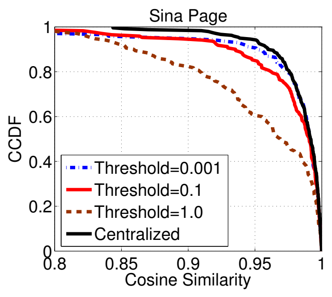
The similarity of two vectors ranges from -1 to $+1$. The lower bound means they are exactly in the opposite direction, while closer to $+1$ indicates they are more likely to be pointing in the same direction. We ran the distributed clustering algorithm for several Web pages and computed the similarity between each sample and its final centroid belonging to the same cluster. We show the results in Fig. 4.5 for three pages. For the distributed algorithm, we show the similarity results using different thresholds. As a benchmark comparison, we also use the centralized k-means clustering method and compute the same similarity values. We use the Matlab k-means module[2] for centralized clustering, running it 100 times and choosing the optimal result (i.e. the one with the smallest sum of “point-to-cluster-centroid distances”).

From the results shown in Fig. 4.5:

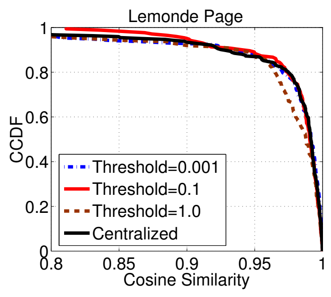
- Most of the samples have high similarity with their centroids even using higher threshold value (e.g., 1.0).
- Distributed clustering using larger thresholds normally has smaller similarities.
- With small threshold values such as 0.1 or 0.001, our distributed clustering already achieves a CCDF distribution very close to that of the centralized method.



(a) google page



(b) sina page



(c) lemonde page

Figure 4.5: Similarity of Clustering Results by Centralized and Distributed Methods in CCDF (clustering on first 4-hour Web page browsing data)

We also give more details of the similarity evaluations in Tab. 4.2. In this table, instead of computing each individual sample’s similarity with its centroid, we show the average similarity of all samples in each cluster with all final centroids. In this case, we can evaluate both intra and inter similarities for distributed clustering. For centralized clustering, we still select the optimal one from 100 runs. From Tab. 4.2, we see that:

- Samples are more similar with the centroid in their own cluster than with others, e.g., the diagonal direction representing the intra-cluster similarity always has a higher value. This proves the reasonableness of the results of our distributed algorithm.
- Distributed and centralized clustering methods have quite close similarity observations for both intra-cluster and inter-clusters, which makes us more confident in using the distributed algorithm as an approximation of the centralized one.

From the evaluation works of this part, we showed that (i) the distributed algorithm works well and can be an approximation to the centralized method; (ii) the required parameters for this distributed algorithm are also easy to define. In the following sections, we will use some case studies involving distributed clustering in a time-series manner.

[2]http://www.mathworks.fr/fr/help/stats/kmeans.html

Table 4.2: Cosine Similarity Comparison between Distributed and Centralized Clustering for Google Pages (**4-hour** data, threshold=**0.1**, other tested pages and more experiments of distributed clustering have similar results)

		Centroids			
Test Scenario	*ClusterID*	*Cluster1*	*Cluster2*	*Cluster3*	*Cluster4*
Distributed	Cluster 1	**0.95**	0.90	0.93	0.82
	Cluster 2	0.92	**0.98**	0.94	0.96
	Cluster 3	0.97	0.93	**0.99**	0.86
	Cluster 4	0.81	0.94	0.85	**0.96**
Centralized	Cluster 1	**0.99**	0.87	0.89	0.95
	Cluster 2	0.85	**0.97**	0.96	0.85
	Cluster 3	0.90	0.98	**0.99**	0.93
	Cluster 4	0.92	0.82	0.86	**0.94**

4.3.2 Case Studies

Since our emulated browsing experiments were made across multiple vantage points within the same period, it therefore allows us to compare performance across different homes. In this section, we pick up some Web pages and and apply our method in a time-series manner. We will make some interesting observations to illustrate the diagnosis and analysis.

4.3.2.1 Home Network Comparisons

We first pick the `google` Web page since it is the most well-known and popular one. Due to the fact that google domain servers can be normally considered to be close to the end user, and also well-engineered for their own network connectivity, it is thus reasonable to investigate the experiences with the google domain as indications of their **local network** comparisons. We ran our distributed clustering by time-series for google pages in different homes and show the results in Tab. 4.3. Unlike Chapter 3, we do not show the raw metrics in a boxplot, instead, we only show the centroids as a representative for each cluster and the corresponding user counter distribution. This is because such information is all that each user can acquire from the distributed clustering; as we will also discuss, this is enough for comparisons and diagnosis at a certain level. From the results shown in Tab. 4.3, we see that:

- The values of TCP for HOME2 are normally the worst. From 20:00 to early morning of the next day, the centroids for most of the google pages in HOME2 keep decreasing, e.g., from clusters 3, 4 of Tab. 4.3(a), to cluster 4 of Tab. 4.3(b), finally, clusters 3, 4 of Tab. 4.3(c).
- HTTP is around 15 ms higher than TCP for most clusters, and this kind of observation occurs for all the homes, therefore, we consider it as the average latency for preparing the data at the server side.
- Parts of the browsing for the EUR network have pretty small DNS delays, which indicates the DNS response from the local server cache.

This performance comparison for the google Web page clearly shows the differences between the local networks of the different homes. EUR (and NAP) normally have the lowest round trip delay to the google domain, while HOME2 has the worst. A time-series analysis for the clustering results also reveals the peak hours for the Internet use of the potential inhabitants of HOME2. As a consequence

Table 4.3: Distributed Clustering (threshold=0.1) for All Google Search Pages (HOME2 is the initiator)

(a) 20:00–00:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	***HOME2***	*NAP*	*TOT.*
Cluster 1	1.0	18.32	34.92	30	0	2	0	32
Cluster 2	22.55	21.39	39.93	26	2	3	37	68
Cluster 3	4.67	**115.43**	98.10	4	0	**20**	1	25
Cluster 4	44.20	**79.93**	96.23	1	34	**14**	0	49

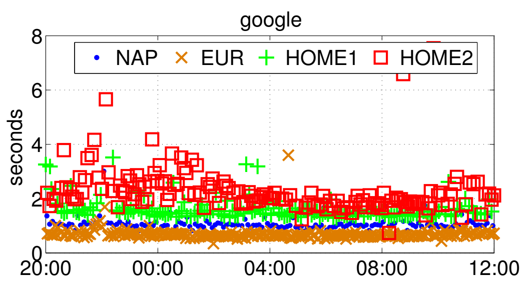
(b) 00:00–04:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	***HOME2***	*NAP*	*TOT.*
Cluster 1	26.13	19.25	34.63	21	0	0	0	21
Cluster 2	1.38	24.12	38.79	43	0	12	37	92
Cluster 3	17.18	44.92	64.21	2	34	11	1	48
Cluster 4	47.91	**81.35**	95.65	0	6	**16**	0	22

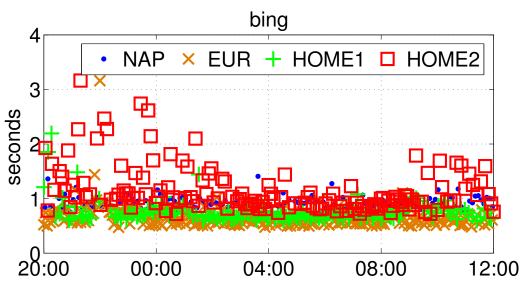
(c) 04:00–08:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	***HOME2***	*NAP*	*TOT.*
Cluster 1	21.02	20.21	32.63	20	0	0	38	58
Cluster 2	1.27	19.28	30.84	46	0	6	1	53
Cluster 3	31.08	**54.84**	70.14	0	38	**16**	1	55
Cluster 4	69.94	**53.54**	69.04	0	2	**12**	0	14

of these observations, Fig. 4.6(a) shows the overall page performance in a time-series. We see that before 2:00 am in that figure, HOME2 has a relatively larger time for the page load; after that time, the page load time decreases and becomes more stable in its variance. As a complement, we also show the page load time in a time-series for the `bing` Web page in Fig. 4.6(b). Similarly to the google page load time in Fig. 4.6(a), we also observe a higher load time variation for the same time period.



(a) Time-Series Page Load Time for Google Pages



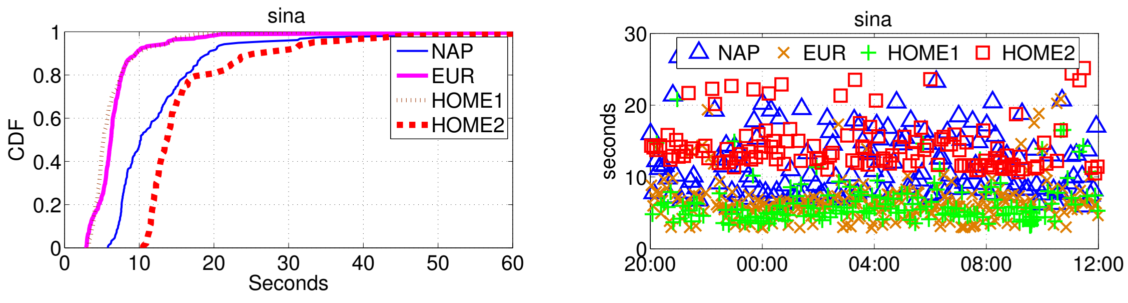
(b) Time-Series Page Load Time for Bing Pages

Figure 4.6: Page Load Time at Different Homes (starting at 2012-12-07 20:00)

4.3.2.2 Impact of DNS Resolution

The previous section shows an example of the network comparison of different homes. Here, we use a Chinese domain, `sina`, to illustrate a different scenario.

The CDF plot for the page load time is shown in Fig. 4.7(a) and the time-series plot is in Fig. 4.7(b). From these results, we see that HOME2 clearly performs worse than the others in all time periods, i.e., the page load times for HOME2 are always larger than 10 seconds, while EUR and HOME1 perform the best; After more detailed investigation, NAP browsing is mainly limited by the client side and we do not discuss such limitations in this chapter.



(a) CDF Page Load Time for Sina pages (b) Time-Series Page Load Time for Sina Pages

Figure 4.7: Time-Series Page Load Time at Different Homes (starting at 2012-12-07 20:00)

When using distributed clustering, the results are shown in Tab. 4.4:

- DNS delay centroids are normally large for all the homes, e.g., 300 ms to 500 ms.
- TCP round trip delays for different homes are quite diverse. All the HOME2 browsing samples are always located in a higher TCP delay cluster, e.g., around 400 ms at cluster 1 of Tab. 4.4(a); however, for other homes, all the TCP round trip delays to the Web servers are much smaller, e.g., around 100 ms.
- If we turn to the HTTP delays, we find that HOME2 still has the largest delay, but the HTTP delay for HOME2 more or less keeps consistent with the TCP delays; while for other homes, the HTTP delays are much higher than the TCP.

Based on these findings, we suspect that the Web servers contacted by these home users may be completely different, which leads to the diversities for TCP round trip time and server side response behavior. To investigate this in detail, we randomly chose one single browsing session from each home, and clustered each individual TCP delay for that Web page. The results are shown in Tab. 4.5. Note that, unlike the previous notation, we use lower case "tcp" in Tab. 4.5 to indicate the individual TCP delay instead of the average value. As in the previous results in Tab. 4.4, the individual TCP delays for HOME2 are still around hundreds of milliseconds, while for the other homes, most of them are in cluster 4 where the centroid is less than 30 ms.

From these results, we can clearly see the round trip delay differences between the homes. Although there are differences in their local networks, as we discussed in the `google` case, those differences are not on the order of hundreds of milliseconds, so we can infer that the performance difference for the `sina` domain Web page is due to the DNS server resolution: For NAP, EUR and HOME1, the DNS servers can return IP addresses with closer distances such as CDN servers; while HOME2 DNS server provides IPs which may be further away (e.g., in China).

A further check by IP organization confirms this inference: Tab. 4.6, lists top-5 contacted server IPs for different homes by the `/24` subnets. We use maxmind [22] to query organizations that an IP address belonging to. From Tab. 4.6 we can clearly see that most of the contacted servers (e.g. nearly 90%) for

Table 4.4: Distributed Clustering (threshold=0.1) for Sina Pages

(a) 20:00–00:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	*HOME2*	*NAP*	*TOT.*
Cluster 1	383.21	**444.89**	**480.69**	0	0	**32**	0	32
Cluster 2	227.74	***82.89***	***221.66***	46	32	0	33	111
Cluster 3	533.76	116.23	235.77	12	2	0	4	18
Cluster 4	738.42	1395.21	409.76	3	0	5	1	9

(b) 00:00–04:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	*HOME2*	*NAP*	*TOT.*
Cluster 1	357.06	225.09	381.75	0	11	0	1	12
Cluster 2	297.69	**447.61**	**511.37**	0	3	**33**	0	36
Cluster 3	520.42	713.34	559.73	1	1	5	0	7
Cluster 4	288.58	***78.13***	***204.64***	64	24	0	39	127

(c) 04:00–08:00

	Final Centroids (ms)			User Distribution				
clusterID	*DNS*	*TCP*	*HTTP*	*EUR*	*HOME1*	*HOME2*	*NAP*	*TOT.*
Cluster 1	439.35	**471.10**	**468.06**	0	0	**17**	0	17
Cluster 2	292.01	**391.00**	**442.71**	0	1	**16**	0	17
Cluster 3	266.73	***89.11***	***230.41***	41	38	0	29	108
Cluster 4	516.34	***75.99***	***199.80***	25	2	0	10	37

Table 4.5: Distributed Clustering (threshold=0.1) of Individual TCP Delays for a **Single** Sina Page

	Final Centroid	User Distribution			
clusterID	*tcp* (ms)	*EUR*	*HOME1*	*HOME2*	*NAP*
Cluster 1	549.10	0	2	**15**	0
Cluster 2	435.00	3	8	**11**	6
Cluster 3	305.24	7	14	**27**	5
Cluster 4	***28.75***	**57**	**47**	0	**55**

NAP, EUR, HOME1 users come from content provider company *Panther Express* and *CDNetworks*[3]; meanwhile, all the contacted servers for HOME2 user are from Chinese telecommunication operators, e.g., over 90% objects are downloaded from a single IP subnet which belongs to *ChineNet Guangdong Province Network*.

4.4 Conclusions

In this chapter, we proposed a distributed method for the Web browsing performance analysis. This method is based on a distributed clustering algorithm and inspired by the discussions of Chapter 3. Compared to Chapter 3, it is improved in that (i) it reduces the complexity of the measurements by heavy packet level capturing; (ii) it avoids any raw data exchange leading to scalability problems. Our results in this chapter have also shown that:

- The distributed methodology can get similar results as the centralized one.

[3]http://www.cdnetworks.com/company/global-network/. Actually, these two companies are merged in 2009 according to: http://www.cdnetworks.com/news/cdnetworks-merges-with-panther-express/

Table 4.6: TOP5 Contacted Server Subnets for All Sina Pages in Different Locations

Locations	*IP*	*Hit(%)*	*Org.*
NAP.	**77.67.3.x**	**89.1**	**Panther Express Corp**
	203.90.242.x	1.8	Newsky Internet Limited
	173.194.35.x	1.5	Google
	61.129.72.x	1.4	China Link Network Technology Limited
	12.130.132.x	0.7	(UpToDate)
EUR.	**91.202.200.x**	**65.4**	**CDNetworks network**
	93.188.131.x	**25.7**	**Panther Express Corp.**
	61.129.72.x	1.4	China Link Network Technology Limited
	219.142.78.x	0.7	Jing'an Center Idc,Chaoyang District, Beijing,P.R.
	203.90.242.x	0.7	Newsky Internet Limited
HOME1	**93.188.128.x**	**52.7**	**Panther Express Corp.**
	174.35.6.x	**9.7**	**CDNETWORKS**
	174.35.7.x	**8.8**	**CDNETWORKS**
	174.35.4.x	**7.9**	**CDNETWORKS**
	174.35.5.x	**5.5**	**CDNETWORKS**
HOME2	**58.63.236.x**	**92.1**	**ChinaNet Guangdong Province Network**
	61.129.72.x	1.5	China Link Network Technology Limited
	180.149.134.x	1.4	ChinaTelecom Group Beijing Ltd,Co
	203.90.242.x	0.8	Newsky Internet Limited
	218.30.66.x	0.7	CHINANET IDC center

- Choosing the parameters for the distributed algorithm is easy and the results converge readily.
- Through some case studies, we showed that even with a limited knowledge of the other neighbors, distributed clustering can still be useful in performance analysis and diagnosis.

However, as a direction for possible future research, it would be desirable to develop models based on the clustering results that automatically deduce the cause of performance problems.

Conclusions for Part I

In Part I, we first introduced the basic measurement platform for our studies in Chapter 2. Due to the Web browser is the closest component for the end user's Web surfing, therefore, our platform is browser-based, while different other components can be added such as packet capture etc. By the browser level measurements, we can easily obtain knowledge of end user perceived performances.

In Chapter 3, we proposed to use clustering methodology to reveal the underlying relationship between QoE and QoS. We have shown that user's subjective feelings can somehow keep consistent with the objective measured metrics; we have also shown that such clustering technique can be used as the collaboration method between multiple homes.

Based on this finding, we extended the clustering work in Chapter 4 to work in a distributed way. We set up a prototype to perform the distributed clustering process among different home users. Although each user can only have limited knowledge such as centroids and counters, we could still show several interesting insights by comparisons among different homes such as local network problems or default DNS server selection problems.

Part II

Troubleshooting Web Sessions in the Wild

Overview of Part II

From previous part of the thesis, we use the "**human-analysis**" method to study the Web performances. we particularly explained Web pages with poor performances. In Part II, we improve the diagnosis method into the **automatic** way, and propose our model to troubleshoot poor performances for wild browsing.

In Chapter 5, we describe the design of the tool and its evaluations. We try our best to make the tool easy to use, non-intrusive and accurate in measurement.

In Chapter 6, we illustrate our model for the diagnosis. We first introduce the performance limitation scenarios, then, highlight the cases that we are going to focus in this thesis. Based on these discussions, we define our algorithm to automate Web performance diagnosis. As we will show in this chapter, our model is only based on passive measurement from end user's wild browsing.

Our results from the controlled experiments show how we choose the thresholds for our model; and the results from long term deployments in several homes reveal multiple causes that can potentially affect the Web page performances.

CHAPTER 5
Design of the Tool

"*You cannot solve a problem from the same consciousness that created it. You must learn to see the world anew.*"

Albert Einstein

5.1 Architecture Description

So far in this thesis, we have presented different methodologies in the analysis of the Web performance. We see that our analysis is based on a browser plugin which can easily combine user perceived performances. In this chapter, we extend it for the wild user deployment, and present **FireLog**, a tool and methodology to quantify Web page load performance and diagnose the limitations. We will show its implement architecture, illustrate the key measured metrics and also evaluate its own properties.

5.1.1 Client Side

The FireLog tool is composed by two different parts: client side engine for measurement and server repository for diagnosis. The client-side part is a plugin to Firefox, which can be easily integrated into the end users' browser; While the user are surfing their Web pages, the plugin will record a list of metrics (described later) and periodically transfer them to FireLog server located in EURECOM. A more detailed architecture for client side capturing engine is depicted in Fig.5.1. From this figure, we see that it contains multiple modules across different levels of measurement. Some of them are similar with previous chapters while additional ones are added to adapt the wild user's browsing preferences:

- **Object monitoring**: since the basic unit of a Web page is the individual objects, then, the basic module in the client capturing engine is the object monitoring module. We use `Javascript listeners` to monitor each object event: whenever an object is initiated by the browser, we will create an array to keep track of all such object's activities. Those activities include the object downloading status such as DNS IP resolving, TCP connecting, HTTP-query waiting and data receiving. Those are the key status periods when an object is retrieved from the Web server.

Besides monitoring the downloading activity, we also record each object byte and the TCP socket that is used to download current object. We also note that for the objects that are completely from browser cache, there will be no networking activities, in that case, we will label the cached object and record only its bytes information.

To be able to group the individual object into the real browsed page level, each time a new object is initiated, we will query its "parent" tab/window and bind the real browsed page with it.

- **Page monitoring**: In order to capture user's perceived Web performances, we use page-level `listeners`. Currently, we record two popular page-level events referring to page `rendering start` and `fully loaded`. In case the browsed page is abandoned by the user before one of these events is triggered, we will only record the page abandoned time.

- **User feedback**: Besides the objective measurement discussed in the above, we also enable user's inter-action. Similar as previous Part I, a button icon is created at the corner of the browser for the user expressing their dis-satisfaction with the browsed pages. Whenever the button is clicked, we will make an annoyance-tag to current page to indicate user's impatience, meanwhile, all the objects that belong to this browsed page and start before the clicking event will be labeled as "annoyance" as well.

  Since the following wild deployments are long-term evaluation for real users, therefore, we do not force our users providing such click action. Currently, we only make the functionality working, and our analysis in the following chapter still relies on the objective metrics.

- **Privacy policy**: Concerning user's privacy issues of our client side measurement engine, we make two policies to deal with it: **first**, user can freely enable/disable the capturing engine with an option button. While it is disabled by the user, all the events listeners will be removed and nothing will be recorded; **second**, all URL information related to user's browsing history, such as browsed page `URL`, HTTP `Host` headers etc. will be hashed (e.g. by SHA-1) into random strings.

- **Data dispatching**: After a given Web page is finished browsing by the end user, we shift the corresponding data to the collector. We define some *key browsing events* by the user as indicators for end of browsing:

  i) We add browse tab listener to detect a tab-removing event. whenever a given browser tab is closed, we assume that user has already finished browsing the page corresponding to that tab.

  ii) For each existing browser tab, we also make a final timeout (e.g. 1 hour) threshold in order to avoid the opened Web page by long absence end user. When the timeout is triggered, the corresponding data is assumed to be expired and dispatched to the collector.

  iii) If the user continues browsing a new page in an existing browser tab, we assume that previous Web page in the same tab is expired.

  iv) Finally, a global window listener is used to detect closing event of the whole browser window. In case such event is triggered, all the data corresponding to the window are assumed to be expired and dispatched to the collector.

  Due to the *single-thread* nature of `Javascript` and avoiding parallel page execution that blocked by data dispatching, we use **asynchronous** HTTP request for data transmission of the former three cases; for the last one case of the whole window is closed, since there is no parallel rendered page in that browser window and to avoid data loss, we use **synchronous** HTTP request for the data transmission, in this case, the whole browser application can only quit when the data transferring is finished. More over, local memory is also released after after the data dispatching.

- **Others**: Besides the above main goals of the FireLog capturing, we generate a random number for each client when it is installed successfully. Such number is persistent along the whole usage of FireLog and it's also transmitted with the normal data to the collector. In this case, we can always distinguish each unique client even its public IP is changed. Besides the previous capturing capabilities, we also use API provided by Fathom [50], and trigger list of system calls such as `ping` RTT to gateway, CPU or memory status, etc. Currently, we only make it functionally working while not adopt these measurements in the diagnosis model in the following.

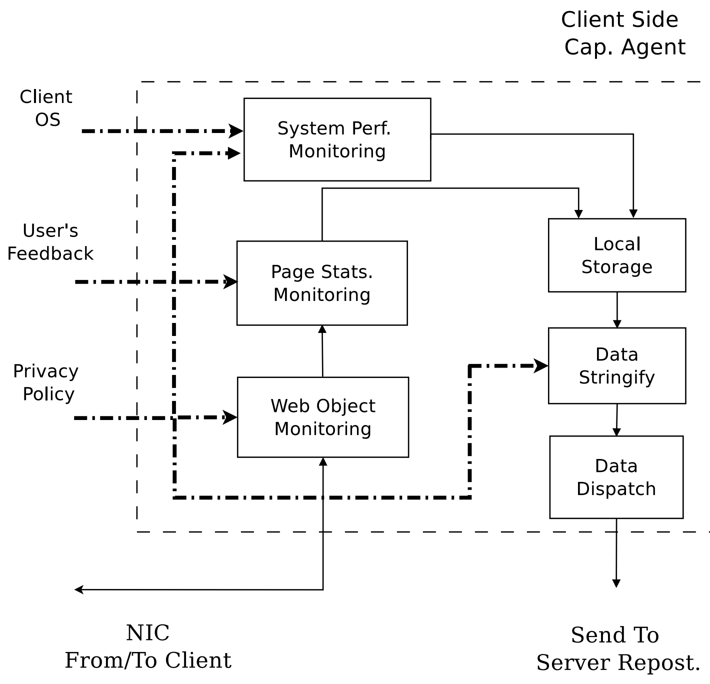


Figure 5.1: Client Side Capturing Engine

5.1.2 Server Side

We set up a server inside EURECOM as the repository for data collection and the diagnosis. As is shown in Fig. 5.2, we use Apache/PHP as front-end that accepts measurement data from the clients, we then transfers the raw data into a PostgreSQL database and assign unique ID per Web page. Index is also created to the corresponding table for efficient query. We implement the diagnosis module (discuss in Chapter 6) in the PL/PgSQL language, and as a complement for debugging, We use the standardized HTTP Achieve (HAR) format [17] to visualize each individual browsed Web page.

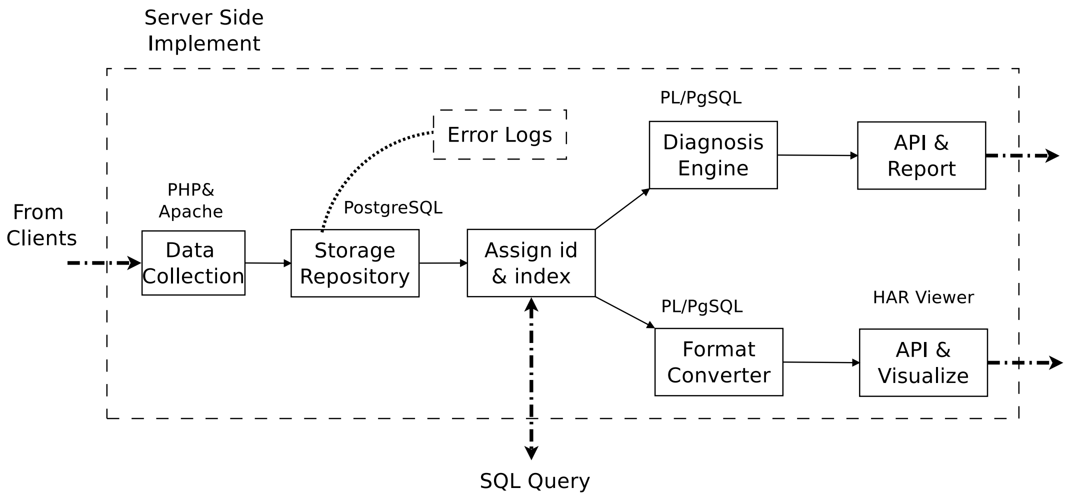


Figure 5.2: Server Side Architecture

5.2 Metric Illustration

As it turns out, modern Web browsers provide a rich set of events useful for our task, which can be captured by FireLog and used to derive the metrics of interest [25]. The key metrics that we will focus are similar with Chapter 4, here, we repeat them in Fig. 5.3:

When a given Web page is accessed by the user, the browser will start to fetch from Web servers the objects that make up that Web page. In this case, there will be different status events appearing in the browser. In Fig. 5.3, the downloading activity is started at t_1 with a DNS name resolution. We measure the time elapsed between the DNS query (t_1) and its response (t_2) as the **DNS delay** (dns=$t_{1,2}$). During such a period, a "`looking up`" text message appears in the browser's status bar. After the DNS lookup, at t_3, which corresponds to sending a SYN packet, a "`connecting to`" text message appears in the status bar until the client receives the SYN/ACK packet (t_4), We refer to this time interval as **TCP connecting (or handshake) delay** (tcp=$t_{3,4}$). Whenever the browser detects that its TCP connection is established, it will immediately change its status (t_6), "`waiting for`" appears in the status bar, and an HTTP query ($t_6^{'}$) is sent. While we are waiting for the HTTP response data from the Web server, several things can happen at: The Web server can either directly send back the data ($t_8^{'}$), or first send back the TCP ACK ($t_7^{'}$) and then return the data. However, at browser level, we can only capture a browser status event that will be triggered at t_8 when receiving the first data. We define $t_{6,8}$ as the total **HTTP query delays** ($http$=$t_{6,8}$). After a successful HTTP response, the browser keeps downloading the object data from Web servers until it is finished at t_9. Besides the metrics just introduced, we can also measure at the browser metrics such as page load time, total number of objects downloaded and total number of bytes downloaded.

5.3 Tool Evaluation

Since FireLog measurement is based on user's browser, therefore, it is necessary to evaluate the tool itself. We do this evaluation work on two different aspects: **browsing overhead** and **time accuracy**.

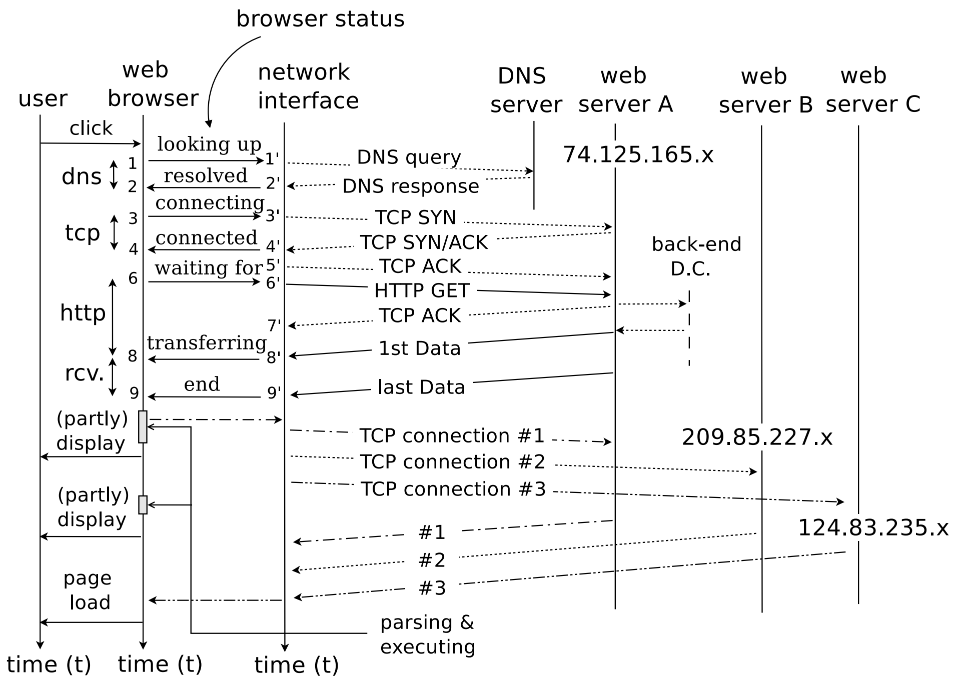


Figure 5.3: Main Metrics Used by FireLog.

Browsing Overhead: We first evaluate the overhead in terms of resource consumption and page loading time. We pick up some popular Web sites in different categories such as `News`, `Computer`, or `Shopping` listed by Alexa[1], and choose other similar tools Firebug [9] and Fathom [50] for comparison. We create different Firefox profiles with separate tool instrumented and also another profile with none of the tools as a performance baselines. For each of our selected page, we keep browsing it 30 times in each Firefox profile and use the average value for each evaluation metric. Fig. 5.4 shows the overhead results for both memory/CPU usages, and `full load` event delays. To make it comparable across different platforms, we plot the CPU overhead with `user` time plus `system` time. From those results, we see that FireLog capturing engine introduces almost nothing impact on the overhead comparing to an empty Firefox browser; while Firebug consumes the most which is similar with the findings in [50].

Timestamp Accuracy: The second evaluation step is for the timestamp accuracy. Since our methodology is based on passive measurement, then, it is important for the accuracy of the measured metrics. To evaluate this, we compare time drift between some key events recorded by FireLog and packet level trace (e.g. by tcpdump). We use the same Web sites and show timestamp drift during TCP handshake and HTTP query phases in Fig. 5.5. From those results, we can see that timestamps measured at both levels have almost negligible drift between each other: for the outgoing direction (e.g. SYN or GET packet sent), only around 3% measurements in SYN packets have 1 ms drift; while for incoming direction, both phases have a bit larger drift (over 3% measurements have larger than 1 ms). We have also tested this timestamp issue with other randomly Web sites browsed by wild home users (describe in the followings), and the results are similar as Fig. 5.5.

[1]www.alexa.com

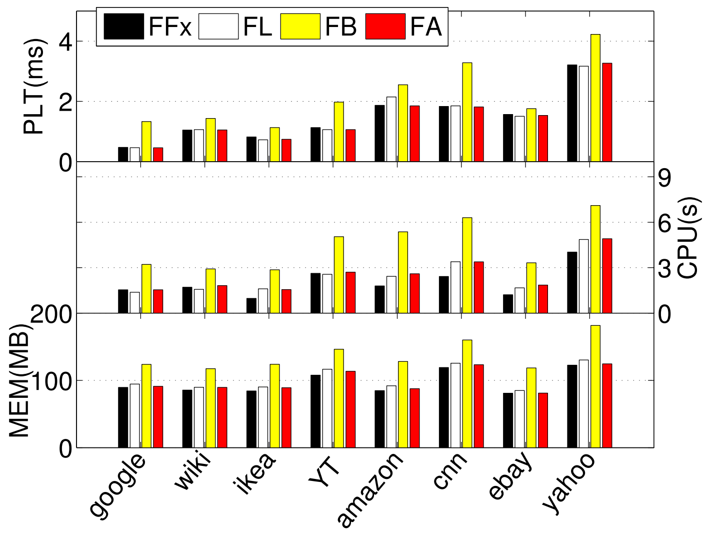
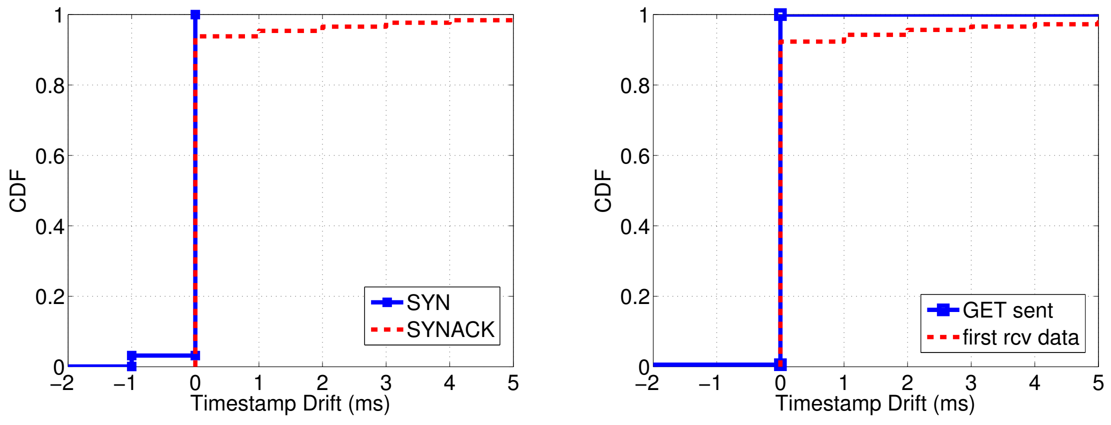


Figure 5.4: Overhead Evaluation (FFx: only Firefox, FL: FireLog, FB: Firebug, FA: Fathom)



(a) TCP handshake phase ($(t_3 - t_3')$ and $(t_4 - t_4')$ in Fig. 5.3) (b) HTTP query phase ($(t_6 - t_6')$ and $(t_8 - t_8')$ in Fig. 5.3)

Figure 5.5: Evaluation of Timestamp Accuracy (other key events such as end of data transferring are similar. For all the results, we only keep the timestamp granularity in the order of milliseconds)

CHAPTER 6
Root Cause Analysis of Web Performance

"Computers are useless. They can only give you answers."

Pablo Picasso

6.1 Introduction

In previous chapter, we have shown FireLog, a tool for Web performance diagnosis for the wild home users. In this chapter, we first illustrate our model for the troubleshooting, validate it in a controlled lab environment, and then, apply it on some wild user deployments for long-term measurement.

Fig. 6.1 illustrates the underlying behavior when browsing page: the main object usually comes first. After that, the Web browser can parse the page structure and load all the objects referred to in the Web page. In order to reduce download times, parallel connections can be also used. After the page content is completely downloaded and rendered, the `full load` event is fired by the browser and the status of the Web pages becomes **fully loaded**. Although modern Web browsers can trigger other object downloads even after the page status is fully loaded [64], in this chapter, however, we do not consider those cases and focus on the ones that have occurred before page fully loaded.

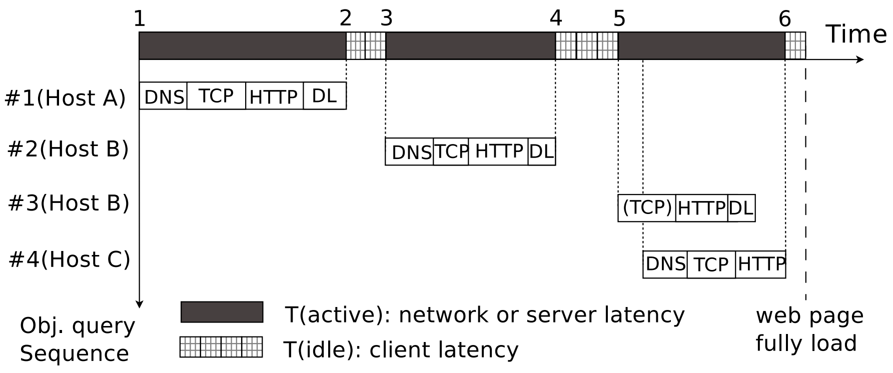


Figure 6.1: An example to show the downloading/rendering of a Web page containing four objects hosted on three different Web servers.

6.2 Diagnosis Scheme

Since a large number of steps need to be executed to completely render a Web page and a number of components are involved in generating transmitting and rendering the content. As is shown in Fig. 6.2, these components are (i) the PC of the client, (ii) the local access link, (iii) the remaining part of the Internet, and (iv) the servers. A slowdown at any of these components will affect the page load time. The goal of this part is to identify which of these components bears the major responsibility for the slow Web page load.

We are well aware that the "overall picture" is more complicated and that, other factors may affect page load time as well such as the Web page size itself in terms of number of objects or total bytes.

Also, we focus on the performance degradation problems while ignoring connectivity issues that have been the focus of other – complementary – tools such as WebProfiler [35] Netalyzr [72].

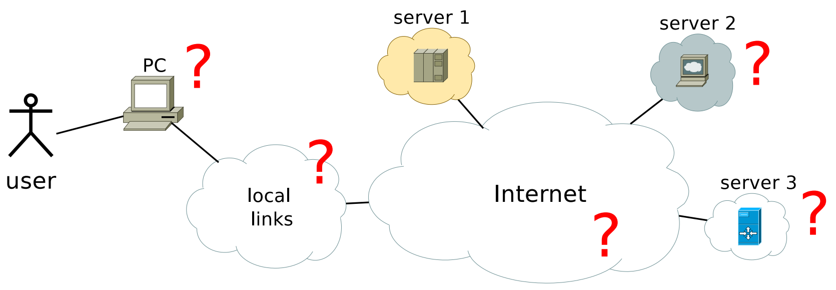


Figure 6.2: End to end path

6.2.1 Proposed Heuristic

Based on the above limitation discussions, we describe our proposed diagnosis heuristics in this section.

6.2.1.1 Main Scheme

Algorithm 1 shows the global diagnosis scheme, and here, we describe the main ideas. We check which of the components in Fig. 6.2 makes the main contribution to a slow page load and we proceed as follows:

Client side diagnosis: lines 5-9 are used to diagnose client side causes. We defined the `C.App.Score`, which captures the fraction of idle periods compared to the total time (see also Fig. 1). A high `C.App.Score` is an indication that overloading of the client PC could take the major responsibility for the long time page rendering.

Network and server side diagnosis: In case there is no client side anomaly, we now check the quality of the communication between the client and servers, which comprises both, the network path and the server response times. We first use an empirical threshold th_{ms} in line 12 to check whether the average http delay is too high. If this is the case, we use the limitation score `Serv.Score` to further distinguish between network causes and server side ones. Lines 13-15 show the diagnosis for server side causes,while lines 16-19 correspond to the network side diagnosis.

Algorithm 1 Web Page Diagnosis Scheme

```
Input: current Web page ($\mathcal{P}$), page load time ($PLT$), start ($ts^i_{start}$) and end ($ts^i_{end}$) downloading
    timestamp for each object, each object HTTP query delay ($http^i$), each Web server TCP connect-
    ing delay ($tcp^{ip}$), current page downloaded object number ($N$) and bytes ($B$).
Output: Web page limitation cause
 1: function WebDiagnosis
 2:     $Idle \leftarrow null$
 3:     $C.App.Score \leftarrow null$
 4:     $Serv.Score \leftarrow null$
 5:     for all objects $i \in \mathcal{P}$ do
 6:         $\sum Idle \leftarrow \textsc{TotalIdle}(P, ts^i_{start}, ts^i_{end})$
 7:     $C.App.Score \leftarrow \frac{\sum Idle}{PLT}$
 8:     if $C.App.Score \geq th_c$ then
 9:         return client side limit
10:     $HTTP \leftarrow \frac{\sum http^i}{N}$
11:     $TCP \leftarrow \frac{\sum tcp^{ip}}{\#ip}$
12:     if $HTTP \geq th_{ms}$ then        ▷ either server side, or network problems, empirical threshold
13:         $Serv.Score \leftarrow \frac{HTTP}{TCP}$
14:         if $Serv.Score \geq th_s$ then
15:             return server side limit
16:         else
17:             $tcp^{ip}_{base} \leftarrow \textsc{GeneratePerfBaseline}(dataset)$
18:             $\mathcal{T} \leftarrow$ current time $\pm 5$ minute                ▷ time window
19:             $Res \leftarrow$ NetwDiagnosis($\mathcal{P}, \mathcal{T}, tcp^{ip}_{base}$)
20:         if $Res \neq null$ then
21:             return $Res$
22:     if $N \geq th^{'}_{size}$ or $B \geq th^{''}_{size}$ then
23:         return page size limit
24:     return unknown                           ▷ no performance anomalies found
```

Other factor diagnosis: In case no previous abnormal behaviors are found by the heuristic, we finally check the page property itself in line 22. We use two empirical thresholds for both object number and bytes to achieve that.

6.2.1.2 Network Case

As is discussed previously, for the network causes, we try to narrow down whether it is the local access or the "wild" Internet. To that purpose, we use measurements for different servers made in a predefined time window. Details are shown in Algorithm 3. In order to be able to conclude the network degradation for a connection to given server, we compute baseline performances for all the servers. Algorithm 2 shows the details of how to extract the baseline where the basic idea is just to choose a lower bound (e.g. currently use 10%th percentile) value for each server in a `/24` subnet. We do this aggregation by the IP prefix to accumulate more samples for each group and make the estimated baseline more robust.

Algorithm 2 Network Performance Baseline Generation

```
Input: whole dataset for a single user ($dataset$)
Output: network performance baseline for each IP subnet
 1: function GENERATEPERFBASELINE($dataset$)
 2:     if baseline data $tcp^{ip}$ exist for this $dataset$ then
 3:         return all $tcp^{ip}$                                        ▷ do nothing
 4:     $BaseList^{ip} \leftarrow null$
 5:     for all objects from $ip$ subnet do                              ▷ IP/24 prefix as subnet
 6:         $tcp^{ip}_{base} \leftarrow \min(tcp^{ip}_{10\%th}, http^{ip}_{10\%th})$
 7:         insert $tcp^{ip}_{base}$ into $baseline$ result table
 8:     return all $tcp^{ip}_{base}$
```

Algorithm 3 Network Diagnosis Scheme

```
Input: current Web page ($\mathcal{P}$), current time window ($\mathcal{T}$), network performance baselines ($tcp^{ip}_{base}$).
Output: network limitation cause
 1: function NETWDIAGNOSIS($\mathcal{P}, \mathcal{T}, tcp^{ip}_{base}$)
 2:     $\mathcal{U} \leftarrow null$                                    ▷ array for current page perf. degradation
 3:     $\mathcal{V} \leftarrow null$                                    ▷ array for recent perf. degradation
 4:     $\Delta \leftarrow null$                                         ▷ temp. variable
 5:     for each $ip \in \mathcal{P}$ do
 6:         $\Delta \leftarrow tcp^{ip} - tcp^{ip}_{base}$               ▷ network degradation
 7:         insert $\Delta$ into $\mathcal{U}$
 8:     if $mean(\mathcal{U}) \leq th_{ms}$ then                         ▷ no network anomaly
 9:         return $null$
10:     for each $ip \in \mathcal{T}$ do
11:         if $tcp^{ip}_{base} \leq tcp^{google}_{base} + th_{ms}$ then  ▷ closer IP
12:             $\Delta \leftarrow tcp^{ip} - tcp^{ip}_{base}$           ▷ network degradation
13:             insert $\Delta$ into $\mathcal{V}$
14:     remove $min$ and $max$ values from $\mathcal{V}$                 ▷ filter outlier
15:     if $\mathcal{V}$ is not diverse enough for its IP samples then
16:         return $null$
17:     $F_1 \leftarrow \frac{mean(\mathcal{U})-mean(\mathcal{V})}{stddev(\mathcal{V})}$   ▷ coincident with others
18:     $F_2 \leftarrow \frac{mean(\mathcal{V})}{mean(\mathcal{U})}$     ▷ local degradation contribution
19:     if $F_1 \leq th_{F1}$ or $F_2 \geq th_{F2}$ then
20:         return local network
21:     else
22:         return wild internet
```

For the network diagnosis in Algorithm 3, we can divide it as following steps: lines 5-7 pick up the contacted servers of current page and check their network performance degradation; lines 10-13 do similar degradation checking, but also choose connections that can be also included by other pages. The idea behind is to pick up the sharing information by different connections (belonging to diverse subnets), in case similar network degradation is discovered, the cause is probably due to the common links among those different servers which is expected to be closer to the client side (e.g. local network links). The tricky part is shown at line 11 meaning that we only pick up recently contacted servers that are relatively closer to the client. In this case, network degradation values are more useful to detect local network problems. As we see at line 11, we use `Google` as a reference[1]. We use Google, because they make great effort to place proxies close to the clients in order to cut down the latency.

Lines 14-15 are used to filter outliers, and check for the diversity of our recently selected servers. In order to make the diagnosis more robust, while keeping enough samples, currently we only filter out the *minimum* and *maximum* values. To guarantee the diversity of these servers, currently, we check whether the number of distinct subnets for those servers is large enough (e.g. ≥ 5 distinct subnets).

We finally use two criteria shown in line 17 and line 18 to identify a local network problem. F_1 is to check whether the current page experiences a network degradation that also coincides (is experienced) by all the "near-by" connections; while F_2 checks whether "local causes" contribute mostly to the current page download degradation. As is shown in line 19, if any of these two criteria holds true, we consider it as local network causes; otherwise as wild Internet.

As we can see from these discussions, our heuristic needs certain amount of thresholds for the diagnosis. We summarize them in Tab. 6.1 and we will motivate the exact choice of the values later in Sec. 6.2.2.

Table 6.1: List of thresholds used in the heuristic.

threshold	*description*
th_c	Threshold to check client side limitation case.
th_{ms}	Empirical threshold to define abnormal higher delay for both HTTP and TCP degradations.
th_s	Threshold to split server side limitation from network limitations.
$th_{size}^{'}$	Large page size threshold in terms of number of objects.
$th_{size}^{''}$	Large page size threshold in terms of total download bytes.
th_{F1}	Threshold to check current page network degradation with other domains.
th_{F2}	Threshold to check whether local network degradation contributes most for current page domains.

6.2.1.3 Flow-chart Interpretation

As a high level interpretation, we show the flow-chart in Fig. 6.3 which corresponds to all the pseudo-codes of previous section.

As we can see from this figure, our root cause analysis is *tree-based*, therefore, the order of the tree branches also impacts the final result. In our diagnosis scheme, inspecting of total client side idle time is done first to guarantee the downloading process is progressive enough by the browser. If we get a page that are rendered smoothly and progressively by the browser, we then check the dynamic latency factors. We put impact factors related to Web page itself in the last step, since it is obvious that dynamic delay factors correlate more to the whole Web performance, and we will also show the results in Sec. 6.3.3.

We mainly focus on the connections to the Web server while ignore the DNS server behaviors. As will be discussed in Sec. 6.3.3, DNS factors normally contribute much less for the whole delays. In fact, we can still include the DNS diagnosis in our main scheme. One proposal is shown in Fig. 6.4 where DNS is checked **before** TCP connections. In case we observe anomalous DNS behaviors, we can classify the cause as *DNS problem*. There are different ways to define DNS as anomalous, one of them could be the method that we used in Chapter 2, where a weight value is computed as total

[1]We consider HTTP request `Host` headers containing "`google`" as key word to be pointing to the Google domain.

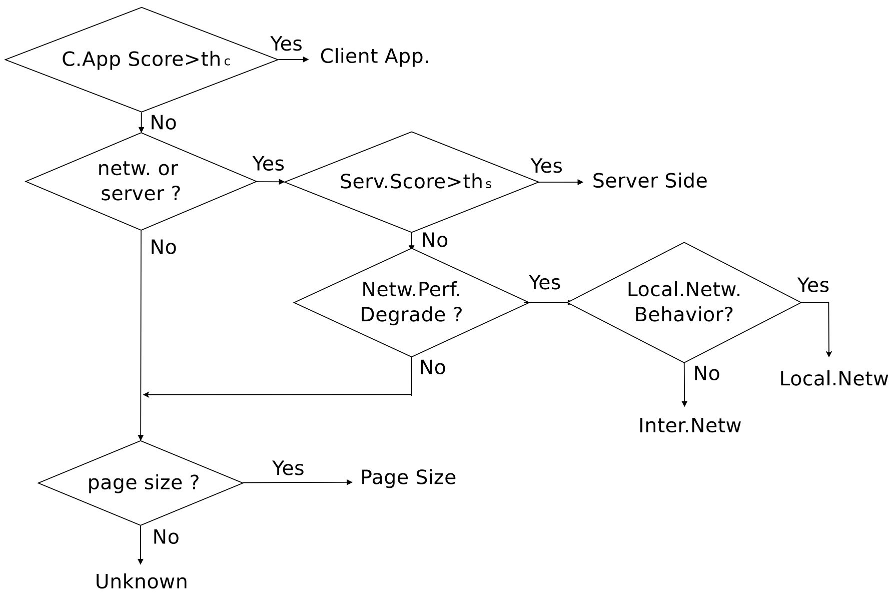


Figure 6.3: Diagnosis Scheme in Flow-chart.

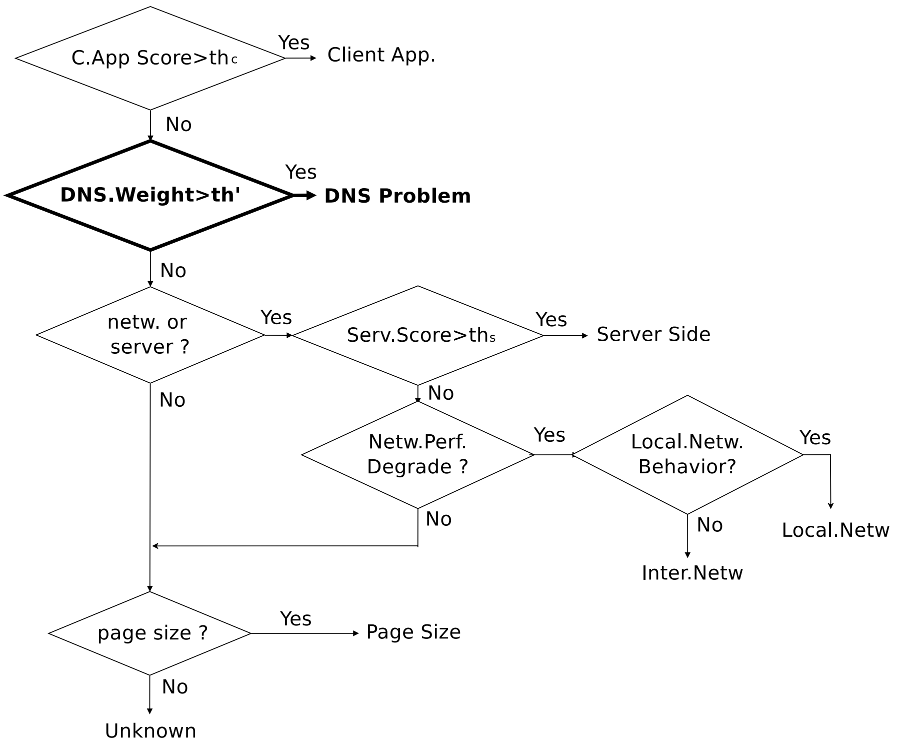


Figure 6.4: Diagnosis Scheme with DNS Taken into Consideration.

DNS delay divided by all downloading delays. As shown in Fig. 6.4, if such weight is higher than a pre-defined threshold, we can consider the whole Web page performance is limited by DNS delays.

There might be other special cases for limiting the Web page rendering, such as the case shown in Sec. 2.3.2.3 where client side proxy exists between client and Web server. In this case, as shown in

Fig. 6.5, we can update our diagnosis scheme by checking all the TCP connecting delays (e.g. all connections of port 80). In that figure, we compute the standard deviation of the TCP connecting delays, if such value is smaller than a threshold, we can consider that a *mid-box* like the HTTP proxy exists, which can potentially limit page downloading.

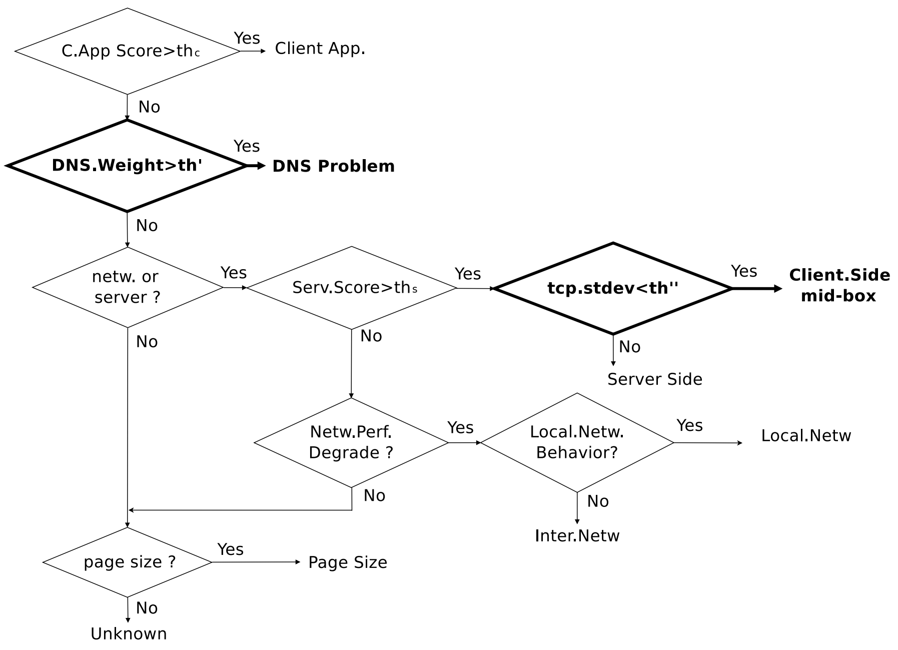


Figure 6.5: Diagnosis Scheme with Client Side Middle-box Taken into Consideration.

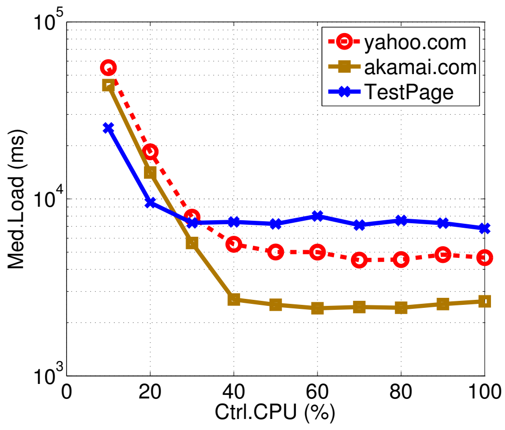
However, we would say that there is no one single model that covers all the performance causes and network configuration scenarios, therefore, in the following sections, we only perform our rules according to the basic version discussed in Sec. 6.2.1 and Fig. 6.3.

6.2.2 Tuning of Threshold

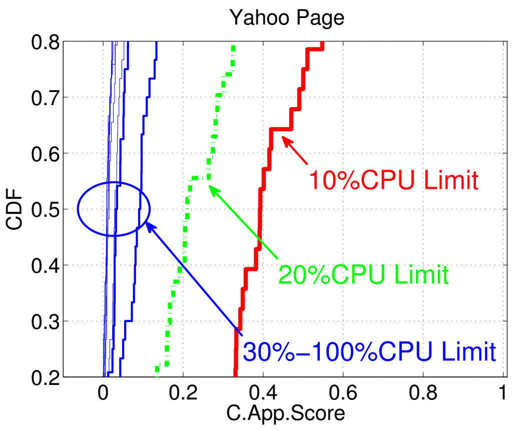
As we can see from the above discussions, our approach needs to define quite a few thresholds. To calibrate these thresholds, we have in most cases done multiple controlled experiments in the lab. Here, we illustrate how we went about.

To set th_c (line 8 of Algorithm 1), which is needed for the client side limitation case, we set up a PC in the lab and browse a list of popular Web pages 30 times each. We use a tool named CPULimit[2] to limit the maximum allowed CPU usage for Firefox browser and observe the browsing performance. We compute the `C.App.Score` for all these test scenarios and report the result in Fig. 6.6. From the result in Fig. 6.6(a), it's obvious to see that, browsing performances are severely limited by the end client side under lower resource scenarios. Fig. 6.6(b) shows the computed `C.App.Score` for Yahoo page: when the maximum allowed CPU usage by Firefox from 30% up to 100%(no limit), such score values are close to 0; while for 20% and 10% CPU limitations, the score values are spited apart. e.g. for the median case, 20% CPU allocation has the score around 0.2, which means 20% time duration of the whole page rendering is idle for the object downloading activities. Fig. 6.6(c) and Fig. 6.6(d)

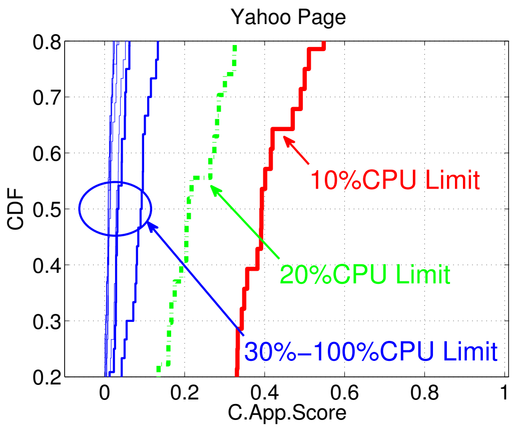
[2]http://cpulimit.sourceforge.net/



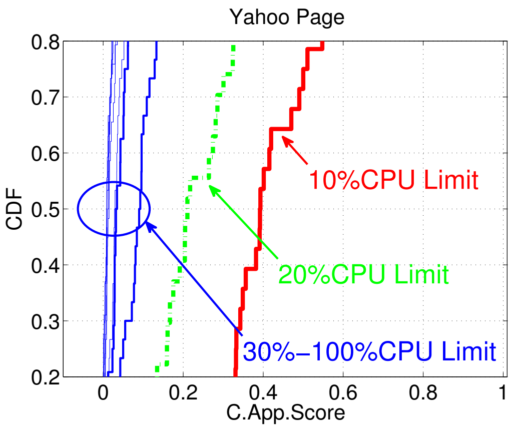
(a) page load time under different limitation scenarios



(b) www.yahoo.com page



(c) www.akamai.com page



(d) local test page with many images

Figure 6.6: Controlled Experiments to Infer Client Limitation Threshold (th_c)

also show the results for other Web pages such as Akamai and a local Web page with full of images, we can observe similar behaviors for the `C.App.Score` as Yahoo case. Therefore, in our following diagnosis, we make the client side limitation threshold $th_c = 0.2$.

Next, we need to set the threshold th_{ms} to identify high delays (e.g. line 12 in Algorithm 1, line 8 in Algorithm 3). In our current version, we manually set $th_{ms} = 100$ ms. Moreover, such value is also used to separate closer servers from more remote ones (e.g. useful in line 11 in Algorithm 3), the reason being that in 2011 and 2012, the minimum RTT from France to US and east Asia were about 136 ms and 271 ms respectively [20].

To define the threshold th_s (line 14 in Algorithm 1), which is needed to separate *network* and *server* side causes, we also set up a controlled experiment as shown in Fig.6.7, where different client PCs are connected through a shared bottleneck. One PC is used for Web browsing while another keeps downloading files to generate a competing traffic at the bottleneck. Fig. 6.8(a) provides a 1-minute snapshot of the `ping` RTT between the client and its gateway PC which shows that random delay around 100 ms always exist. We browse several popular Web pages for long time and explore a range of values from 1.0-4.0. We then compute the fraction of sessions classified as local network limitation.

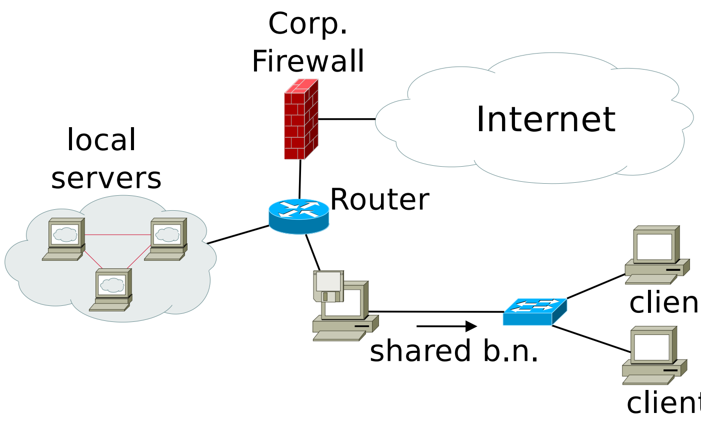
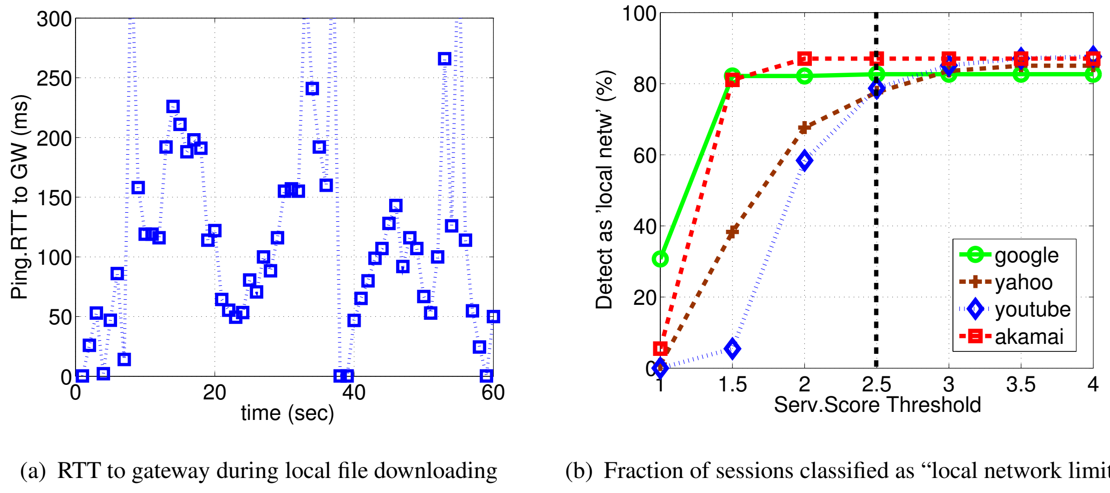


Figure 6.7: Lab Experiment Setup



(a) RTT to gateway during local file downloading (b) Fraction of sessions classified as "local network limit"

Figure 6.8: Controlled Experiments to Infer Threshold (th_s)

Fig. 6.8(b) shows the results by applying different th_s values: from 1 to 2, the "correct" classifications for all the Web pages are increased sharply. When increasing the threshold from 2.5 and so on, it is almost stable and converged around 80% for all the pages. From this result, we set 2.5 as a proper threshold.

For the diagnosis of local network limitation, we use two criteria in line 19 of Algorithm 3. For these threshold values, we empirically choose $th_{F1} = 1$, and $th_{F2} = 0.5$. Finally, we use recent results by Google[3] based on billions of existing Web pages to choose the thresholds for large page size (line 22 in Algorithm 1). We use the 90-th percentile values for the page object number and total bytes which are 86 and 663.19 KBytes respectively.

[3]https://developers.google.com/speed/articles/Web-metrics

6.3 "Wild" Deployment

In the following, we present results of a deployment of our tool at three different home users for a duration of several months each. Two of these users are in French Riviera and the other is located in China. While these results do not replace a careful evaluation of the tool in a controlled lab environment they, however, allow to demonstrate the potential of the tool. While we have no "absolute ground truth", the fact we know the access network characteristics of the three sites and the Web sites browsed allows us to some extend to check the plausibility of the results.

6.3.1 Measurement and Dataset

We select three users that differ in age, education, and geographical location to guarantee the diversity of the browsed Web pages. Two of our users are located in France while another one is in China. A summary of our collected data and results is shown in Tab. 6.2. All these users have accessed a large number of pages. Since our goal is to diagnose Web pages with high load times, we focus on the one with page load time larger than 10 seconds and refer to them as "high load time" ones.

Table 6.2: Collected Dataset from three users.

		Totally Browsed			Web Pages with "High Load Time"		
user	duration	$\#page$	$\#domain$	$\#object$	$\#page$	$\#domain$	$\#object$
$A(FR)$	5 month	3,451	579	501,102	808	247	142,939
$B(FR)$	3 month	1,788	263	87,898	281	114	24,406
$C(CN)$	2 month	3,766	535	317,700	466	183	63,619

6.3.2 Limitation Analysis

Tab. 6.3 shows the main classification results by our diagnosis scheme. We see that for all three users, there are always around 20% pages are limited by the client side. Meanwhile, user A suffers quite a lot from network performance problems, both, local network and wild Internet. We also find for users B and C that around 40% of the high load times are due to the server side. In the following, we look at these results in more detail.

Table 6.3: Limitation Causes for Web Pages with high load times

User	Main cause				
	Client	Server	Local access	Internet	others
A	**21%**	4%	**29%**	**32%**	14%
B	**28%**	**39%**	9%	10%	14%
C	**21%**	**44%**	9%	6%	20%

6.3.2.1 Client Limitation

We first focus on the client side limitations. Fig. 6.9 shows the `C.App.Score` for all these high load time Web pages. We find that in around 80% of the cases for all these users, the `C.App.Score` is quite small, i.e. below 0.2; while for the remaining Web pages, the score takes values up to more than

0.9. Since the curve bends at a value of about 0.2 for `C.App.Score`, we feel comforted in setting the threshold to 0.2.

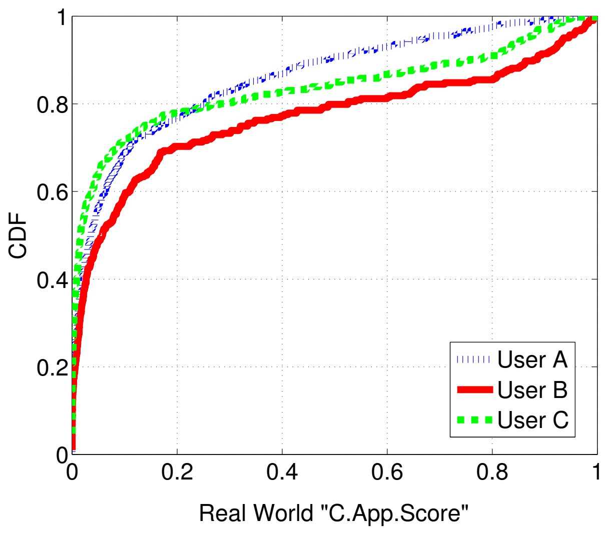


Figure 6.9: Client Limitation in the "Wild"

6.3.2.2 Network Performance

The high page load times of user A are in many cases due to poor network performance. User A lives in a student residence with a shared WIFI link that is frequently overloaded as we can also see if we look at its RTT to Google, which is much higher than for user B. In Fig. 6.10 we see that around 80% TCP RTTs for user B are smaller than 30ms and 90% is smaller than 100ms. Meanwhile, for user A, performance is much worse and half of the values are larger than 100ms.

As we can also see from Tab. 6.3, "wild Internet" is another main limitation reason for high page load times experienced by user A. In this case, we choose Web pages whose high page load are caused by the wild Internet and compare them with the ones where the local network causes high page load times. We look at the fraction of objects that were downloaded from Web servers identified as "further-away". A given Web server IP address is considered as "further-away" if its baseline delay t_{base}^{ip} is $t_{base}^{ip} > tcp_{base}^{google} + 100\ ms$. From Fig. 6.11 we can clearly see that the Web pages where the wild Internet and not the local network is identified as the main cause of a high page load time fetch much more objects from servers that have a higher RTT distance.

6.3.2.3 Server Side Cause

Another major cause for high page load time can be "server-side" factors, which seem to be predominant for both, user B and C.

Here we look at one user to explore this issue for more detail. In Fig. 6.12(a) we plot all the TCP connecting and HTTP query delays of server-side limited pages for user B. We can clearly see that the network delay between client and the Web servers is low, 60% of the TCP delays are less than

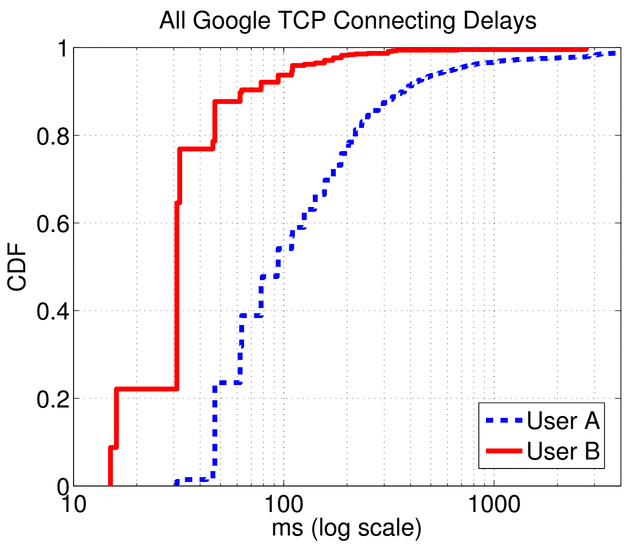


Figure 6.10: Google RTT

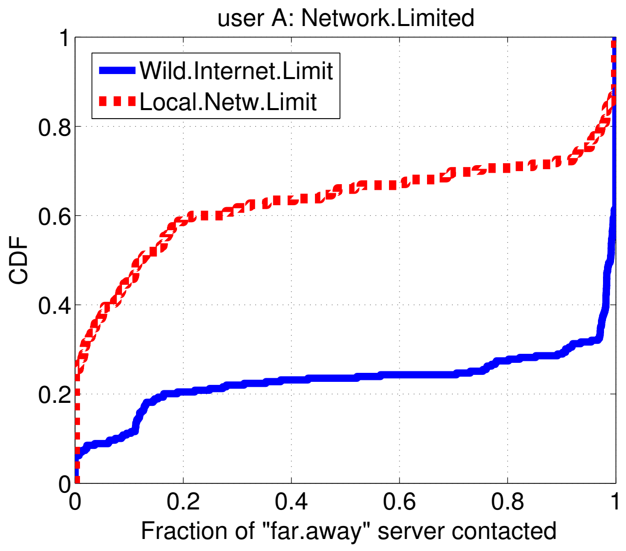
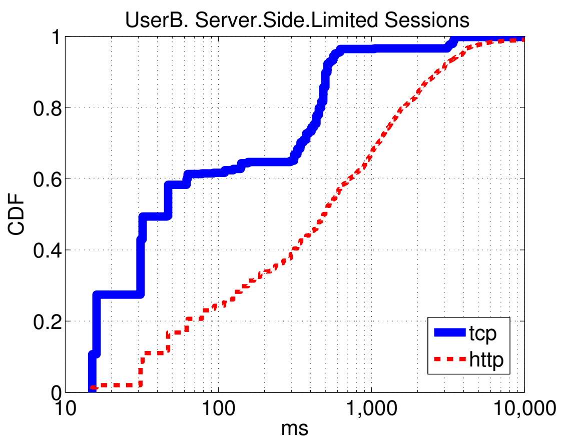


Figure 6.11: All Network Limited Pages for User A.

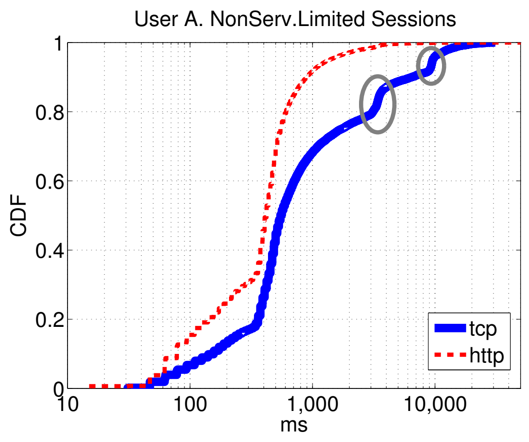
100ms. Since the TCP handshake normally use 3 seconds as its re-transmission timeout (RTO), we also observe very small portion($< 2\%$) of TCP delays with values around 3 seconds, which may due to network loss. However, if we look at the HTTP delays in Fig. 6.12(a), we see much larger values: the median value is already as large as 500 ms. As a comparison, we also in Fig. 6.12(b) plot these metrics for user A, where the server is rarely the cause for high page load times but rather the network. Here the network delays are higher (in distribution) than the http delays. We clearly see some TCP delays around 3 seconds and 9 seconds which are determined by the RTO values.

As a complement to the results shown in Fig.6.12(a), we look at all these server side limited pages and use Alexa to check their country information[4]. Tab. 6.4(a) summarizes top-5 countries. We see

[4]We use Alexa (www.alexa.com) to query home page of them.



(a) User B: Serv.Limit



(b) User A: Netw.Limit

Figure 6.12: Single Object TCP and HTTP Delay Comparisons

Table 6.4: Server Side Limited Web Pages for User B.

(a) top-5 countries

country	*count*
CN	66%
JP	15%
US	10%
FR	6%
IN	1%

(b) top-5 Org. for CN/JP pages

Org.	*count*
CDNETWORKS	18%
NC Numericable S.A.	10%
ChinaNet Guangdong	8%
ChinaUnicom Beijing	7%
Panther Express	6%

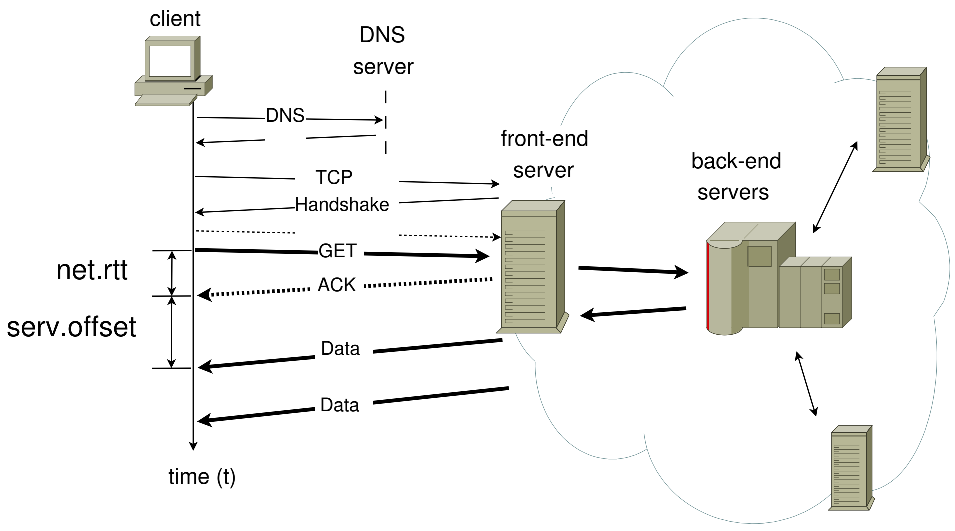


Figure 6.13: Server Side Interaction and the Impact on Client Side Measurement

that over 80% pages are either Chinese or Japanese ones, however, if we look at the contacted servers in Tab. 6.4(b) (by maxmind [22]) we find a high usage of CDN servers: three of them (in bold font) are either well-known CDN providers (CDNETWORKS, Panther Express) or local network operators

(NC Numericable S.A.). Therefore, we infer that for these server side limited pages, as is shown in Fig. 6.13, the client may not directly contact the Web server which holds the real page content. Instead, front-end servers are used to establish the TCP connections. After that, the client sometimes needs to wait for the front-end server fetching real content from back-end content-holders (maybe further-away locations such as China or Japan etc.). Such server communications make HTTP delays measured at the end client longer. Indeed, similar observations are also made by Chen et al. in [47], where they studied the impact of interaction between front-end and back-end servers to `bing.com` Web page.

6.3.2.4 More Discussions of Server Side Interaction

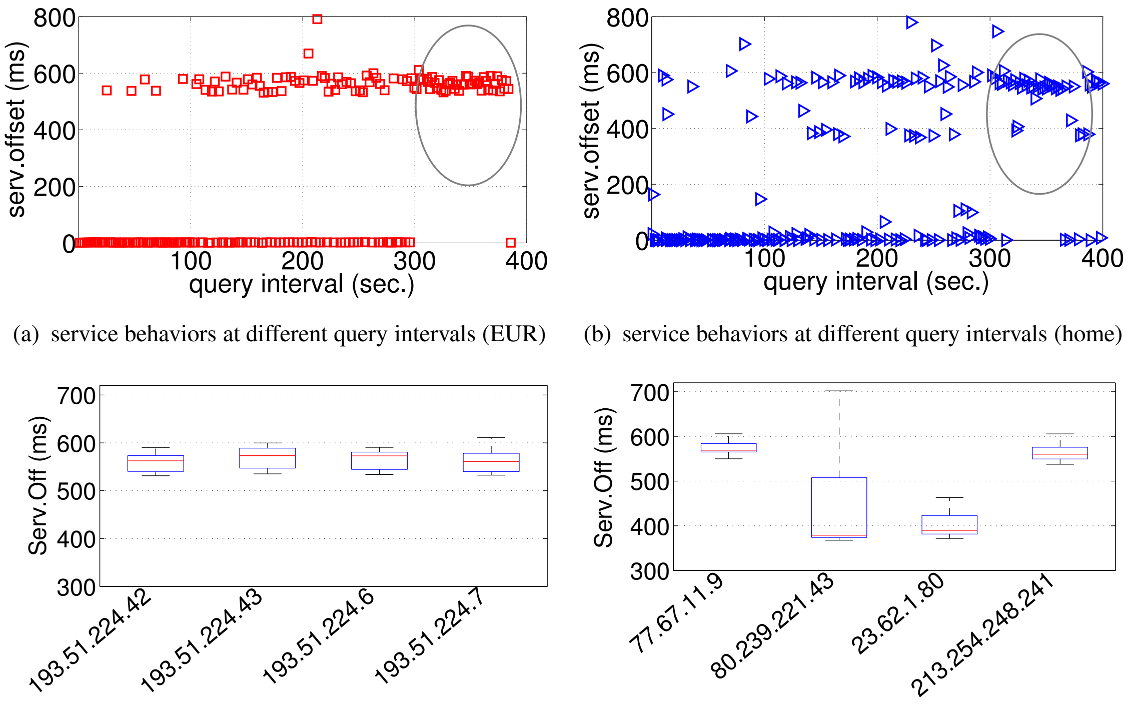
Server side behavior In order to reveal details of server side behaviors, we pick up one domain as an example and perform active measurements. We use the mostly browsed Web domain from all *server-side limited* pages of user B[5]. We set up one PC in EURECOM and another one in another home environment. We use `wget` to periodically query for the main HTML object (few KB large) of the home page of that domain, and meanwhile, we capture the whole trace at packet level. The tricky part in our experiment is that, between each consecutive `wget` query, we wait for some time period and the waiting time is increased by 2 seconds for each consecutive query. To feature server's response behaviors, we compute service offset (e.g. *serv.offset* in Fig. 6.13) based on the packet level trace. Such metric is more precise to characterize server's behavior since it splits network latency (e.g. *net.rtt* in Fig. 6.13) from a HTTP response time. Fig. 6.14(a) and Fig. 6.14(b) show the service offset results for our consecutive queries under different waiting time. From these two figures, we find that, the service offset mainly located at certain ranges around 0 ms, 500 ms and some of them in the home environment are around 400 ms. During small time intervals, e.g. $<$100 sec, most of the server offset is close to 0 which means the contacted server immediately replies with data. When the interval between consecutive queries increases, we have more opportunities to get responses with larger server offset. When the waiting interval is larger than 5 minutes (300 sec), we can see that almost all the responses have service offset in the order of hundreds of milliseconds. This observation implies that, for shorter (e.g. $<$100 sec) waiting time between each consecutive queries, we are likely to receive the data directly from the contacted Web servers (e.g. by the CDN server cache), in this case, we have almost 0 ms service offset; when the intervals are large enough (e.g. $>$300 sec), the front-end server can erase their cache, and whenever it receives the client's query, it is likely to further contact back-end content holder which makes the client observe a large HTTP query delay.

If we also take a closer look at the contacted server IPs in the two measurement points, we find these are completely different. EURECOM network always resolve the object host to `193.51.224.x` Akamai subnets, while the home client mainly contacts four other IPs. We show the measured service offset for different measurement points and server IPs in Fig. 6.14(c) and Fig. 6.14(d). We can see that EURECOM response IPs have more constant and stable service offset; while for the home client, the differences can be as high as 200 ms for different IPs, although they are always within Akamai domain[6].

Impact on user experiences From previous discussion, we observe different behaviors of the Web server for the HTTP queries. Sometimes it may directly fetch from its local cache, otherwise, contact-

[5]For the debug purpose, this user's browsing is recorded in clear text. For the privacy reason, we hide the domain name.

[6]By using Maxmind, all the server IP addresses are in Akamai domain



(a) service behaviors at different query intervals (EUR) (b) service behaviors at different query intervals (home)

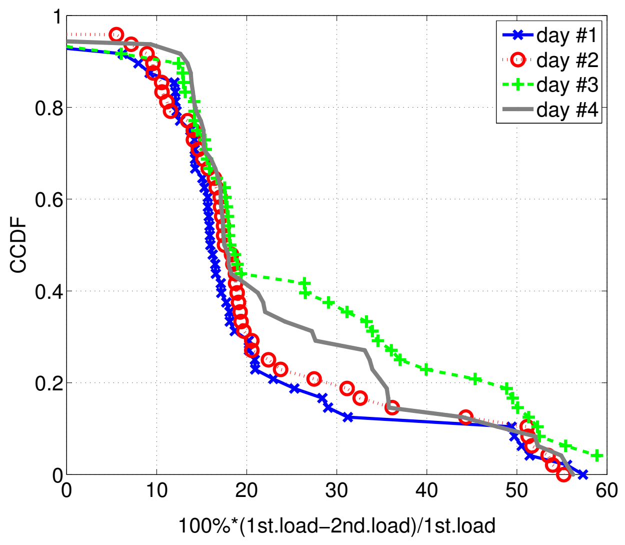
(c) service time for most contacted IPs (EUR) (d) service time for most contacted IPs (home)

Figure 6.14: Main Object Query to Study Behaviors between **Front-End** and **Back-End** Web Servers.

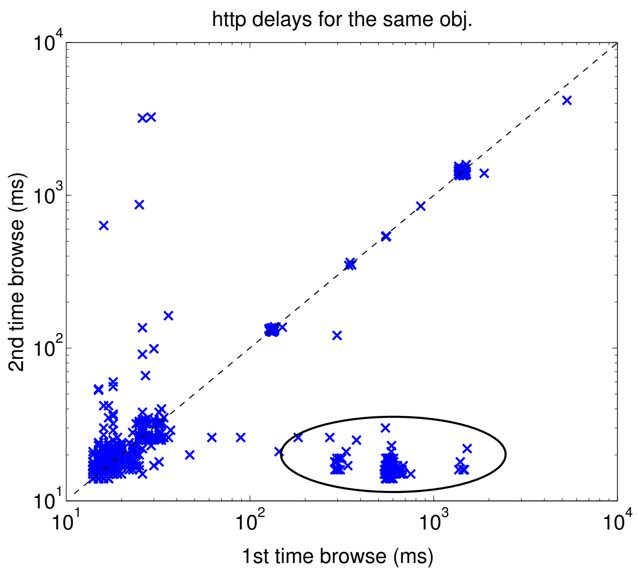
ing back-end servers. In this section, we show how this behavior can impact user perceived performance.

We pick up all the pages with the same domain as used previously, and remove the pages that do not exist any more. After the filtering, we get 50 different Web pages browsed by user B. For each selected Web page, we use the one client PC in EURECOM to automatically browse it twice back-to-back. Before browsing the next page, we stop it for 6 minutes since we observe previously that 5 minutes should be proper threshold for front-end server to erase their cache. By this operation, we can assume that, for first browsing of a given page, front-end server is more likely to communicate with the back-end server; while the second time browsing can make it fetching from local cache. In order to make our results robust enough, we do the experiments in several independent days. To quantify the page performance differences of the **same** page between the two browsing, we compute their relative differences as $\frac{1st.load.time - 2nd.load.time}{1st.load.time} \times 100\%$, and Fig. 6.15(a) reports the results for all these days' experiments in CCDF. We see that, for almost all the page browsing, the second time browsing can always get improvement comparing to the first one. Around 40% pages can have more than 20% improvement in the second browsing, more impressively, 10% Web pages in the second time browsing can even reduce the page load time more than 50%.

As a complement, we use the first day's experiment, and compare the HTTP delays measured by FireLog for the same object within same Web page in Fig. 6.15(b). From that figure, most of the HTTP query delays between the two times of browsing are similar (e.g. along the diagonal of the plot). However, there is also another group (around 10%) of objects where the first time browsing has hundreds milliseconds HTTP response delay while it is only around 20 ms for the second time. We can infer that, this group of objects are fetched directly from the front-end servers in the second time



(a) page load time improvement when some objects are cached at CDN servers



(b) http delay comparison for same object(same Web page)

Figure 6.15: Impact for user perceived performance by the behaviors at Web server side.

browsing without contacting to the back-end servers.

In all the discussions above, we use one particular domain and extra active experiments to show how the underlying behaviors are for the server side interactions and their impact on user perceived performance. We would say that it is impossible to do such experiment for all our measured Web pages or domains, thus, we only choose the most popular one from a single user and demonstrate in details. We show that by using our diagnosis model, *it allows us to discover such interesting domain and reveal details of their server-side interactions*. This is also another way around to validate our methodology in the wild.

6.3.3 Discussion

In this chapter, we presented a methodology to diagnose the cause for high page load times. As we said before, it is impossible to address all possible causes, e.g. DNS delay is not taken into our account in our approach, since we find that – from our measured wild data – DNS delays count much less for the total downloading delays and we show this in Fig. 6.16. We compute the weight metric with the same method used in Chapter 2, which takes total DNS latency divided by sum of all the objects downloading time within a page. We see that for most of the Web pages, DNS delays have small weight values. This can be explained by the fact that (i) wild browsing pages can be linked from similar domains where IP addresses are already resolved by previous connections; (ii) DNS pre-fetching is commonly used in modern browser which can also make it less weighted as shown in that figure.

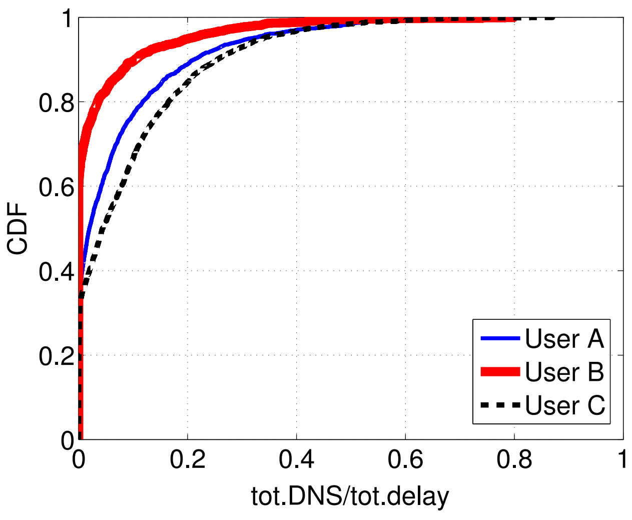


Figure 6.16: DNS Weight for “Wild” Web Pages

Moreover, we focus on the limitation factors of *time variant* metrics such as network delays or server load. While Web page properties such as object number or bytes also have certain impact on the page load time [43]. However, when focusing only on Web page whose load times are high, these static features are less important as indicate the correlations for some of the key metrics with the page load time. Tab. 6.5 reports the results[7] for different limitation causes. We first find that, all the Web page property related metrics such as object number or bytes have much weaker correlation with a given cause for a high page load time than other *dynamic* factors such as TCP or HTTP delays. Also and not surprisingly, we find that, for the client side limited case, page load time strongly depends on the total client side idle time; while for the other limitation scenarios, the total HTTP query delay and TCP connecting delay impact most server limited and network limited Web browsing sessions respectively.

[7]In that table: Nr.Obj and Byte refer to the total object number and bytes including the cached ones. Nr.DL.Obj. and DL.Byte refer to the total download object number and bytes that are not found in the local browser cache. Netw.Limit pages include both local network and wild Internet limited cases.

Due to the use of parallel connections during a page downloading, however, these correlations are not as strong as for the total idle time in the client side limited case.

Table 6.5: Spearman Correlation Between Different Metrics to Page Load Time for Bad Performed Web Pages of All Users

	$\sum Idle$	$\sum dns$	$\sum tcp$	$\sum http$	$Nr.Obj$	$DL.Obj$	$Bytes$	$DL.Bytes$
$Client.Limit$	**0.83**	0.18	0.24	0.39	0.24	0.19	0.18	0.09
$Serv.Side.Limit$	0.08	0.07	-0.02	**0.44**	0.14	0.13	0.11	0.12
$Netw.\ Limit$	0.25	0.32	**0.60**	0.49	0.36	0.38	0.35	0.38

6.4 Comparison With Other Related Work

This chapter defines rules to provide a main cause for a given page. However, there can be situations where not one single reason explains a high page load time, but a combination of several reasons; therefore, as a complement, we use clustering (traditional kmeans) to process our data. We describe each Web page download by a vector of metrics as:

$$[C.App.Score, TCP, HTTP, Nr.]$$

where $C.App.Score$ represents the fraction of idle period during the whole page downloading, TCP and $HTTP$ stand for the average value of the TCP connecting and HTTP query delay respectively, and $Nr.$ is the total downloading object number. Details of steps such as data normalization, number of clusters etc can be found in Chapter 3. We still choose **four** as cluster number.

We pick up all the client limited pages for user A as the example, and do the clustering process. Tab. 6.6 shows the results for the metrics, and we can see that, besides the client side limitations, network performance in cluster 2 and 3 is also worse than other clusters, especially for cluster 3. Pages in cluster 2 also have much larger size. Based on these reasons, we can see from the page load time results in Fig. 6.17: cluster 3 normally has higher page load time; meanwhile, since the network performance and page size in cluster 1 are generally better, page load time in cluster 1 is smaller than others in the distributions in Fig. 6.17. From these results, we see that although poor performance of Web pages can be classified as the the **same** type by our diagnosis model, they can still have different behaviors caused by other reasons (e.g. network factor etc.). in this case, *clustering can be considered as a complement for our diagnosis heuristic*.

Table 6.6: Clustering Results of **Client Side Limited Pages** for User A

	Metric Median Values				
$ClusterID$	$C.App.Score$	TCP	$HTTP$	$Nr.$	$Tot.Page$
Cluster 1	0.33	388 ms	205 ms	56	73
Cluster 2	0.30	**1533 ms**	706 ms	**231**	35
Cluster 3	0.42	**10954 ms**	464 ms	53	14
Cluster 4	0.74	348 ms	190 ms	33	50

6.5 Related Work

The related work can be classified into several categories:

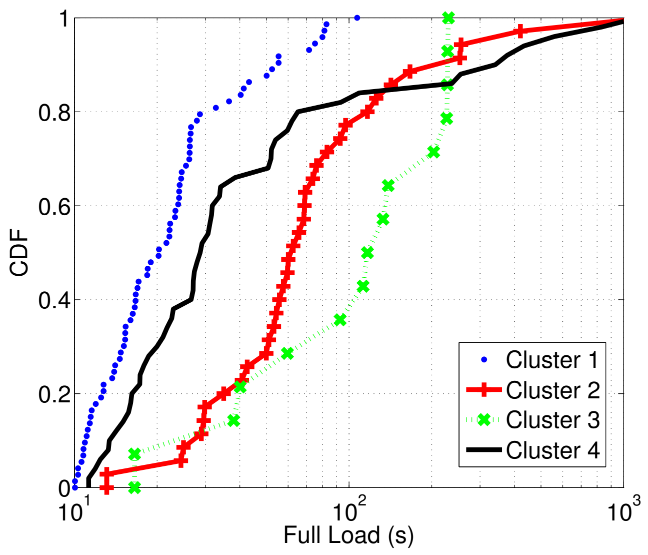


Figure 6.17: Page Load Time in Different Clusters for **Client Side Limited** Pages in User A

Tools: For Web page debugging or monitoring purposes, Web browser itself (e.g. Google Chrome or IE) can integrate developer tools for time-line visualization or Web page debugging [2] [18]. Extra plugin can be also integrated such as Firebug [9], one of the most well known tools for Firefox, which has modules for the page element inspection or activity visualization etc. However, these tools lack a systematic troubleshooting model and may also introduce a significant browsing overhead [50]. Similar tools to directly embed in user's browser also exist, such as Dynatrace [7], SpeedTracer [15] and Fathom [50]. Apart from these browser oriented tools, Another online testbed WebPageTest [31] provides a global distributed platform with diverse Web browsers instrumented for the browsing test.

Troubleshooting: Another category is about troubleshooting model development. For example, Siekkinen et al. in [85] propose a root cause analysis model for TCP throughput limitations for long connections. However, this model does not apply in our case since Web connections are often quite short in terms of the number of packets transmitted, A very recent work that also uses a browser plugin for network troubleshooting is Fathom [50]. However, focus is not the same; Fathom more broadly measures a wide spectrum of metrics that characterize a particular Internet access such as access bandwidth, bottleneck buffer size, DNS behavior. In this sense Fathom is complementary to FireLog since it can be used to further investigate into the reasons of high page load times that FireLog identifies as caused by the local access link.

Web Performance and User Impact: The third group of work correlates Web browsing performance with page properties (e.g number of objects, use of CDNs) [43]. Ihm et al. [64] provide a long longitudinal view of Web performance changes. Nah et al. [75] includes user participation during Web page browsing which shows 10 seconds or more of page load times will lead to user dissatisfaction.

6.6 Conclusions

This chapter presents the FireLog model for Web performance diagnosis. We also deploy the tool on several home users for several months, which allows to collect a large enough data set that provides interesting insights into the diverse limitation categories and the potential of the tool. As a complement, we use extra active measurement for some interesting domains that are discovered by FireLog to illustrate more details of server side interactions and the impact on user perceived performance.

Conclusions for Part II

In this part, we focused on the diagnosis for ordinary home users. We first presented our tool, FireLog, that is composed by client side measurement and server side diagnosis and analysis. We used experiments to show the acceptable features of FireLog, such as low intrusiveness, high accuracy etc.

We then illustrated the potential limitation factors that can limit Web page rendering. We used some examples to show the limitation factors that we focused, namely client side, network side and server side limited etc.

After that, we introduced our diagnosis model for troubleshooting. Although there are multiple parameters to be set ahead, we relied on some controlled experiments to show the reasonability of these parameter values.

After long-term (over five months) deployment at real home users, we have collected enough data set. By picking up the Web pages with bad performance, we found that (i) there is always client side limited across all the users; (ii) traditional metrics that evaluate network behaviors such as TCP round trip time can sometimes be insufficient to explain high page load time, since high portion of browsing sessions are detected as server side limited.

Part III

A New Method For Analysis of the Web Browsing

Overview of Part III

Previous parts of the thesis have addressed different issues related to the Web performances. Among these methodologies, one common problem is that: we have taken the whole Web page as **a single entity**. However, in the real Web pages, it's usually not the case. objects in a Web page normally have different impacts for the whole page performance.

In this part, we develop a new methodology to extract key factors during a page is downloaded. We get the inspirations from "critical path method" and we also try to build such critical path for a given Web page rendering.

There are several scenarios that can benefit from our methodology: (i) performance bottlenecks can be easily found since activities in the critical path are more important to the overall performance; (ii) performance can be easily predicted by updating the deduced dependency graph; (iii) key objects inside a given page can be easily found since the more important for one object, the more frequent it is selected in the critical path under diverse browsing conditions.

We will show in this part the intuition behind our methodology, definitions and also the results for popular Web pages.

CHAPTER 7
Critical Path Method for Web Analysis

"*A project without a critical path is like a ship without a rudder.*"

Daniel B. Meyer

7.1 Introduction

From previous chapters, we have shown different methodologies in the Web performance analysis and diagnosis. However, there are still limitations for these approaches (and also related literatures in this field): 1) Existing approaches are either based on individual TCP connections or group **all** the objects (or connections) belong to a given Web page. 2) Complicated dependency relationships exist between Web page objects, therefore, simple average or sum of all the measured metrics sometimes have flaws.

To overcome these limitations, we propose in this chapter a technique called **Critical Path Method** (CPM) to analyze Web page download performance. We use the idea of finding a critical path among the elements of a Web page to discover *key elements* during Web page rendering. As we will show in the chapter, our method not only allows to explain high page load times but also to study *what-if* scenarios that evaluate the impact of performance enhancing techniques such as using caching etc.

The Critical Path Method was proposed first in 1959 by Kelley and Walker [5]. It was originally meant to solve project management problems related to the scheduling of activites. However, *any project with interdependent activities can apply this method of analysis* [5]. In our case, we can consider the page rendering process as a *project*, where the object downloadings are the *activities* that need to be finished during such project. Hidden dependencies always exist between each other and finding the critical path among them can help us identify key activities. CPM determines the objects in the critical path impact the final Web page load time[1]. In order to find a critical path, according to its definition [5], we need:

1) All downloading activities during the rendering of the entire page.

[1]In [5]: *This process determines which activities are "critical" (i.e. on the longest path) and which have "total float" (i.e. can be delayed without making the project longer).*

2) Inter dependencies between these activities.

3) Time duration for each download activity.

In the following, we will separately discuss these issues.

7.2 Object Download Activities

Multiple object downloads are involved before a Web page can be fully rendered. For each object download, we can identify different steps. We show an example for a single object in Fig. 7.1: When the object is request to be downloaded from a Web server, the browser will first resolve the IP address by a DNS lookup for the corresponding URL at time t_{dns}. After the IP address is resolved, the browser can establish a new TCP connection at time t_{tcp}. As soon as the browser has established the TCP connection, it will send the HTTP query (e.g. HTTP GET message) to the Web server at time t_{http}. If the requested object is valid, the Web server will notify the browser with the specific HTTP status code (e.g. 200) and prepare the transfer of the object data. During this time period, whenever the client detects the requested data started arriving at t_{data}, it will keep receiving the data until the last byte for the object arrives at $t_{data.end}$.

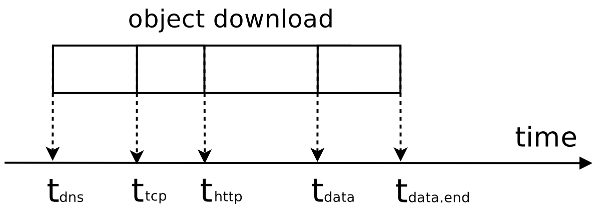


Figure 7.1: Activity Definitions

The situtation is slightly different when the object is downloaded from a Web server whose IP address has already been resolved. In this case there will be no DNS lookups and we can consider t_{dns} as *undefined*; In case the TCP connection has been previously established, when persistent connections are enabled and object downloads re-use existing TCP connections, t_{tcp} is *undefined.* If the current object is not fetched from the local cache, HTTP query and data receiving will be always mandatory. We define the whole downloading activity as “active” period and starting (t_{start}) and end (t_{end}) time for this object are:

$$t_{start} = min(t_{dns}, t_{tcp}, t_{http})$$

$$t_{end} = max(t_{data}, t_{data.end})$$

In the following sections, t_{start} and t_{end} are the basis for our methodology.

7.3 Dependency Discovery

When t_{start} and t_{end} for each object are computed, the next step is to find the inter-dependency between the objects. In this section, (i) we first define which types of dependency we are looking for;

(ii) we then use some examples to illustrate how the download activity will behave when such dependency exist; (iii) based on these intuitions, we finally propose and explain our dependency extraction algorithm.

7.3.1 Dependency Definition

Typical Web pages contain multiple types of objects. Studies show that modern Web pages have become more complicated [64], which also impacts the end-user perceived performance [43]. Therefore, identifying inter-dependencies between the objects can help us identify the performance bottlenecks. In our work, We consider *one object depends on a list of other objects if downloading the former* ***children object*** *can only occur when the downloads of all the latter* ***parent objects*** *are finished.* To be more specific, we consider the following types of dependency relationships:

1) Reference Dependency: If an object in a Web page is triggered by a previous referring object in the same page which leads to this request, we say they have reference dependency relationships. A typical example is the embedded objects that depend on the main HTML file, e.g. when the browser parses the `<img src=.../>` in the HTML page, the corresponding object inside the tag will be requested for download; embedded objects in the Styling Sheet[2] can be also triggered after the Styling Sheet object is fully downloaded [80] [81].

2) Redirection Dependency: In case a new object is loaded as a consequence of a redirection contained in a previous object, we consider these two objects to have a dependency relationship. A typical example is the `google.com` home page from which a user can be redirected to other pages such as `google.fr` or `google.it` according to their language preferences or locations.

3) Blocking Dependency: When a new object is ready to download but is blocked due to a limit on the maximum allowed parallel connections[3], then, the new object can only be started when one of these parallel objects finishes its download. In this case, we consider that such an object has a dependency relationship with the parallel downloaded objects that delayed its own download.

4) Inferred Dependency: This type of dependency can not be explicitly observed. Instead, it is usually inferred from some special behavior. For example, object requests can be triggered by Javascript executions although no explicit information such as HTTP header is available to identify such a dependency relationship.

Based on these descriptions, we will now present our methodology to automatically extract these dependencies from timing and statistic observations among shared experiences.

7.3.2 Dependency Identification

Identifying dependencies precisely between objects in a given Web page is challenging. However, we will show that dependencies can be inferred by observing the downloading activity of each object. For example, to identify the first two types of dependency discussed before, we can rely on some basic HTTP headers: embedded objects that are part of the main HTML will set their `Referer` HTTP

[2]CSS tag {`background:url("http://...") left top;`} can include embedded background images.

[3]Common browsers always have such settings such as recent firefox (e.g. from version 5 up to 17) set their maximum persistent connection per server as 6.

header as the URL of the main file[4]; redirection parental relationships are easy to identify since we can rely on the HTTP response `status` and `Location` headers.

For the third type of dependency as discussed before, since the maximum number of allowed parallel connections can be directly read from the browser settings, the blocking parental relationship can be also easily identified: We update the counter of the parallel connections[5] for each object whenever it starts to download. In case the number of parallel connections reaches to the upper bound, following object that starts *immediately after* these parallel objects can be considered as being blocked by one of the parallel parents.

The most tricky part is *hidden* relationships (the fourth type discussed before) that can not be identified using information that are explicit headers or parameter settings of the browser. In this case, we rely on certain features during object downloading. We first assume that for a given object, all objects that *finish before* its starting time as its parent candidates; then, we remove or confirm such parental relationship from observations of other browsing experiences for the same Web page. For example, If two objects have parental relationship, they can not be running in parallel. In case current object and one of its parent candidates can run in parallel, the parental relationship should be removed. A formalized way to describe the parallelism behavior is shown in Fig. 7.2, we let

$$F = max(t_{end}(A), t_{end}(B)) - min(t_{start}(A), t_{start}(B))$$

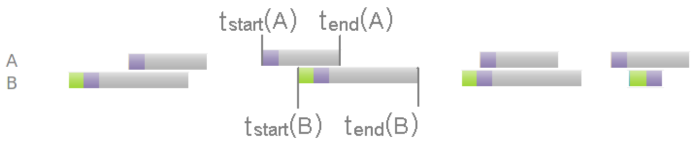
and the downloading durations of the objects are:

$$T_A = t_{end}(A) - t_{start}(A)$$

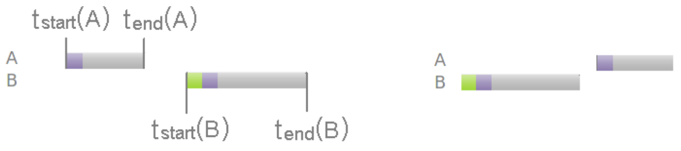
$$T_B = t_{end}(B) - t_{start}(B)$$

Then, we check if two objects are running in parallel or not by

$$F : \begin{cases} < T_A + T_B : & \textit{parallel running} \\ \geq T_A + T_B : & \textit{not in parallel} \end{cases}$$



(a) Cases that Object A and B are Running in Parallel



(b) Cases that Object A and B are Not Running in Parallel

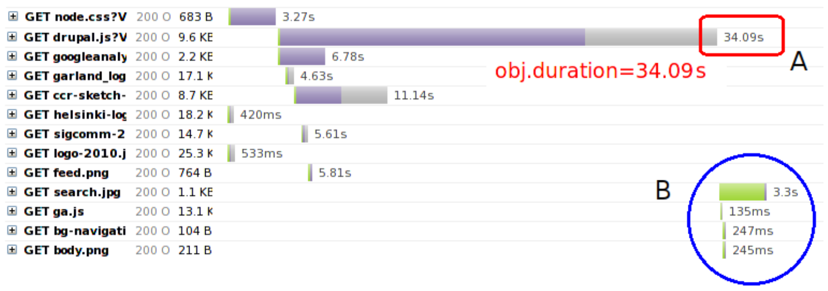
Figure 7.2: Parallelism Checking between two Objects

Another special case between a given object and its parent candidates is that when we observe one of the parent candidate's ending time is much larger than all the other candidates, we say it as a kind

[4]Although such header is optional in the client side settings, as is shown in [64], it is enabled in most Web browsers by default.

[5]There are three types of such settings: `max.persistent.connections.per.host`, `max.connections.per.host`, and `max.connections`.

of “parent pruning”. Parent candidate with largest ending time in this case should be included as the real parent. A simple example is shown in Fig. 7.3: new objects in group `B` are postponed by a single object `A`. Thus, we infer `A` as the parent object for group `B`. Such pruning can happen when, for example, parent object is scripting type which can block object downloading [80] [81].



```
⊞ GET node.css?V    200 O  683 B   3.27s
⊞ GET drupal.js?V   200 O  9.6 KE  34.09s
⊞ GET googleanaly   200 O  2.2 KE  6.78s
⊞ GET garland_log   200 O  17.1 K  4.63s
⊞ GET ccr-sketch-   200 O  8.7 KE  11.14s
⊞ GET helsinki-log  200 O  18.2 K  420ms
⊞ GET sigcomm-2     200 O  14.7 K  5.61s
⊞ GET logo-2010.j   200 O  25.3 K  533ms
⊞ GET feed.png      200 O  764 B   5.81s
⊞ GET search.jpg    200 O  1.1 KE  3.3s
⊞ GET ga.js         200 O  13.1 K  135ms
⊞ GET bg-navigati   200 O  104 B   247ms
⊞ GET body.png      200 O  211 B   245ms
```

Figure 7.3: Parental Relationship Inferred by “Parent Pruning”.

After filtering between a given object and its parent candidates, if there are still candidates existing for a given object, we finally use correlation to infer the parental relationships. Our intuition is that if two objects have parental relationships, parent object ending time should have a strong positive correlation with the starting time of its child object.

As we can see from the above discussion that our dependency extraction is based on several observations. The first three types can be inferred from a single browsing experience; while the fourth type needs other shared experiences from multiple downloads of the **same** web page. Due to the **randomness** in the download latencies, it allows us to define rules to detect possible dependency relationships between each other.

7.3.3 Algorithm for dependency extraction

Based on the above descriptions, we now formalize our algorithm to extract dependencies. Tab. 7.1 summarizes main notations that are used in the algorithms. Note that the key metrics measured to identify dependencies are starting ($t_{start}(i)$) and ending ($t_{end}(i)$) time of an object; $P1(i)$, $P2(i)$, $P3(i)$ are variables to store the number of existing parallel connections when given object i starts.

Table 7.1: Notations for the Algorithm

Notation	*Meaning*
$t_{start}(i)$	Start of downloading time for object i.
$t_{end}(i)$	End of downloading time for object i.
$P1(i)$	Parallel persistent connections per server when object i starts.
$P2(i)$	Parallel connections per server when object i starts.
$P3(i)$	Parallel connections when object i starts.
$ExplicitParent(i)$	Parent object for i that is explicitly identified by HTTP header.
$BlockParent(i)$	Parent object for i that blocks i’s downloading.
$InferParent(i)$	Inferred parent objects for object i.
$ParentSet(i)$	Possible parent candidates for object i.

Algorithm 4 shows the initialization process: we first assign the parent objects that can be easily inferred from basic HTTP headers (line 5 of Algorithm 4); then, all the objects ending before a given object i will be assigned as parent candidates of i (line 7-8 of Algorithm 4).

Algorithm 4 Algorithm Initialization

```
Input: current browsing experience (e.g. $t_{start}(i)$, $t_{end}(i)$, basic HTTP headers)
Output: Initial parental candidates matrix $ParentSet(i)$.
1: function INITIALIZATION
2:     sort all objects by $t_{start}(i)$
3:     for each object $i \in sorted\ objects$ do
4:         $BlockParent(i) \leftarrow null$
5:         update $ExplicitParent(i)$ by Referer and Location headers
6:         update parallel connection numbers for $i$
7:         for all $j$ s.t. $t_{start}(j) > t_{end}(i)$ && $j \notin ExplicitParent(i)$ do
8:             append $j$ into $ParentSet(i)$
```

Algorithm 5 Blocking Parent Extraction

```
Input: $t_{start}(i)$, $t_{end}(i)$, parallel connection numbers for each object, basic browser settings
Output: $BlockParent(i)$
1: function BLOCKINGPARENTEXTRACTION
2:     for each $i \in sorted\ objects$ do
3:         $BlockChildren \leftarrow null$
4:         $BlockParent(i) \leftarrow null$
5:         if $P1(i) \geq max1$ or $P2(i) \geq max2$ or $P3(i) \geq max3$ then
6:             for all parallel objects $J$ at $t_{start}(i)$ do
7:                 sort all $J$ by $t_{end}(J)$
8:                 $x \leftarrow \underset{i}{\mathrm{argmin}}(t_{start}(i))$ s.t. $\forall i \notin BlockChildren$ and $t_{start}(i) \geq t_{end}(J.pop)$
9:                 if $t_{start}(x) - t_{end}(J.pop) < \delta$ then
10:                    $BlockParent(x) \leftarrow J.pop$
11:                    append $x$ into $BlockChildren$        ▷ one object $x$ can be only blocked once.
```

Algorithm 6 Inferred Parent Extraction

```
Input: Candidate matrix $ParentSet(i)$
Output: $InferParent(i)$
1: function INFERREDPARENTEXTRACTION
2:     for $i$ in sorted objects and $ParentSet(i) \neq null$ do
3:         $ParentSet(i) \leftarrow$ PARALLEL$(i, ParentSet(i))$
4:         $InferParent(i), ParentSet(i) \leftarrow$ PRUNE$(i, ParentSet(i))$
5:         if ParentSet(i) $\neq$ null then
6:             $Selected \leftarrow$ STRCORR$(i, ParentSet(i))$
7:             append $Selected$ to $InferParent(i)$
8:             $ParentSet(i) \leftarrow null$        ▷ clear candidate
```

In Algorithm 5, we now check if the parallel connections have reached the maximum number. Whenever the maximum number is reached (line 5 of Algorithm 5), we consider these parallel objects as a group (line 6 of Algorithm 5). If following objects are blocked by this group, they should start **immediately after** completion of any object in the group (line 9 of Algorithm 5). In our current algorithm, we use 10 ms as a threshold (δ in line 9) to determine whether follow-up object is started immediately or not.

Algorithm 7 Sub-Modules that are Used in Inferred Parent Extraction

```
1: function PARALLEL($i$, $ParentSet(i)$)
2:     for $j$ in $ParentSet(i)$ do
3:         for all other experiences do
4:             if $\exists\, i, j$ active in parallel then
5:                 remove $j$ from $ParentSet(i)$
6:     return $ParentSet(i)$

7: function PRUNE($i$, $ParentSet(i)$)
8:     $Parents(i) \leftarrow null$
9:     for $j$ in $ParentSet(i)$ do
10:        for all other experiences do
11:            if $\exists\, j$ s.t. $t_{end}(j) > t_{end}(x) + \alpha\ (x \in ParentSet(i),\ x \neq j)$ then
12:                remove $j$ from $ParentSet(i)$
13:                append $j$ into $Parents(i)$
14:    return $Parents(i)$, $ParentSet(i)$

15: function STRCORR($i$, $ParentSet(i)$)
16:    $Selected \leftarrow null$
17:    for each $j \in ParentSet(i)$ do
18:        query all $t_{start}(i)$, $t_{end}(j)$ from other experiences
19:        remove outliers with much larger $(t_{start}(i) - t_{end}(j))$ values    ▷ e.g. $> \theta$-th percentile
20:        if PEARSONCORR($t_{end}(j)$, $t_{start}(i)$) $> \beta$ then
21:            append $j$ into $Selected$
22:    return $Selected$
```

In Algorithm 6, for each object, if its parent candidates are not null, we query other shared experiences of the same page and check the parallelism (line 3 of Algorithm 6), parent pruning (line 4 of Algorithm 6), and statistical positive correlations (line 6 of Algorithm 6). After this filtering, as shown in Algorithm 6, the remaining inferred parent objects will be assigned (line 7 of Algorithm 6). Algorithm 7 lists the sub-modules that are used in Algorithm 6.

As final step, for each of the non-isolated objects i, we assign its parent objects as **union** of all the discovered parents by previous steps (e.g. $ExplicitParent(i)$, $BlockParent(i)$, $InferParent(i)$). We can see that several parameters (or thresholds) need to be defined. Tab. 7.2 summarizes them with descriptions and the values used. For these parameters, maximum parallel connections are chosen as the default values of recent Firefox[6]. Other parameters are chosen empirically.

7.4 Critical Path Extraction

After the dependency extraction is done, we can represent the Web page downloading activities as a Directed Acyclic Graph (DAG) containing node set V and link set E. Any node in V represents an object of the given Web page. Nodes can be connected through links in E. Each link $i \rightarrow j$ in

[6]These values did not change since Firefox 6.0.

Table 7.2: Thresholds for the CPM Algorithm

Name	*Description*	*Value*
$max1$ (line 5 of Algorithm 5)	maximum persistent connections per server	6
$max2$ (line 5 of Algorithm 5)	maximum connections per server	8
$max3$ (line 5 of Algorithm 5)	total maximum connections	256
δ (line 9 of Algorithm 5)	whether blocked object starts immediately after ending of its parent	10 ms
α (line 11 of Algorithm 7)	end time of one parent candidate is much larger than other candidates	1 sec
β (line 20 of Algorithm 7)	pearson correlation between end time of parent candidates and start of children	0.7
θ (line 19 of Algorithm 7)	threshold to remove some largest distance between $t_{end}(parent)$ and $t_{start}(child)$	80%

E represents a parental relationship between parent object i and its child object j. After the DAG is determined, we define "root" as the object that has no parent node directly pointing into it and "leaves" as the object that have no child nodes.

We then assign values that represent the time delay of each object to the graph. We consider the two types of delays: *downloading delay* and *client delay*. The former refers delays incurred to download the object while the latter refers to client side effect while rendering the content that was downloaded. We use T_i to denote the total downloading delays for object i and $T_{i,j}$ to denote the estimated client side delay from parent object i to child object j. For a given object i, we compute $T_i = t_{end}(i) - t_{start}(i)$; However, computing the client side delay is not so straightforward as we can see from Fig. 7.4: Suppose that object 4 in that figure has three parent objects 1, 2 and 3 and object 4 can only be loaded when **all** its parents are completed. Among those parent objects, 3 is delayed more than the others ($t_{end}(3) > max(t_{end}(1), t_{end}(2))$), therefore, starting time of object 4 is further delayed by the end time of object 3. Thus, we infer the client side delay between object 3 and 4 ($T_{3,4}$) as the time gap between them. However, the processing delay between 1 and 4 ($T_{1,4}$) and 2 and 4 ($T_{2,4}$) are not easy to identify based on solely timing information. In this case, we manually assign such client delays $T_{1,4} = T_{2,4} = 0.01$.

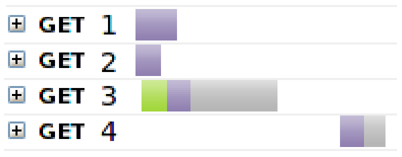


Figure 7.4: Client Side Delay Assignment

When the timings are assigned in the DAG, we use greedy method to check all distances of the paths traversing from *root* object to the *leaves*. Distance of each path is sum of all the time values of the nodes and links belonging it. The path with largest distance from the root to the leaf node is defined as the final **Critical Path** for a whole Web page.

7.5 Experiment Results

7.5.1 Instrumenting

Our current CPM implementation approximately consists of 1000 lines of Python code. We use Firefox as our browser with Firebug [9] to track the object downloading activity as defined in Fig. 7.1. We also store the log file in HAR format for the visualizations[7].

We first show some typical examples by deploying our CPM algorithm. During these examples, we continuously browse selected Web pages with an empty browser cache. In order to emulate different browsing conditions we manually change client machine's out-going packets with potential delays or losses using `netem` toolkit[8]. We configure the out-going packets with four different scenarios: 50ms, 100ms packet delays and 10%, 20% packet drop rate. We keep 60 seconds in each scenario and then go into next one. For each of the selected Web page, we browse it for 50 different times to ensure that there are some shared information for the same Web page to query. Since all the experiments are done within the same day, we observe that Web page contents do not vary a lot. In case dynamic changes in the object Universal Resource Identifier (URI), we match objects by the *levenshtein edit distance*[9] of their URI strings.

7.5.2 Case Study

7.5.2.1 Google Page

We first show the `google` Web page results in Fig. 7.5. As we can see from Fig. 7.5(a) that this page consists 10 different objects including HTML, Javascript and images. By parsing the objects with our CPM algorithm, we obtain the results shown in Fig. 7.5(b):

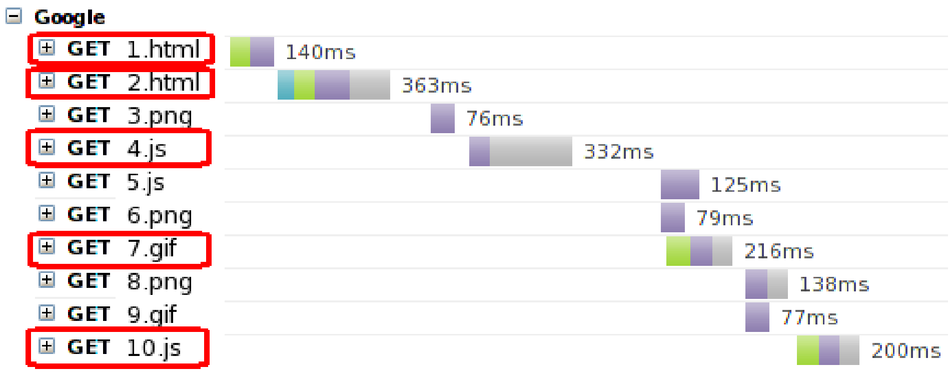
- `1.html` (`www.google.com`) object is the root that initializes the whole Web page.
- After receiving the data of `1.html`, the page is redirected to the main file of `2.html` which is `www.google.fr`.
- When the browser receives enough data from the main file, it starts to render the page content and downloads other embedded objects either in parallel or serial. Therefore, we can see as dashed lines in Fig. 7.5(b) that a direct link pointing from `2.html` to all the other objects.
- According to our observations, throughout all different network conditions for downloading the embedded objects, the Javascript file `4.js` can always block the following object downloading. Therefore we see from Fig. 7.5(b) that there is a direct link from `4.js` to the following objects.
- Groups of objects `5.js`, `6.png` and `7.gif` can be downloaded in parallel, however we find that longer delays of any object in this group will further block following ones. In this case, we see from Fig. 7.5(b) that these three objects have direct arrows to the following child objects `8.png`, `9.gif` and `10.js`.
- We can also see a direct link from object `3.png` to `8.png`, `9.gif` and `10.js`. This may due to the fact that in our experiments, ending time of `3.png` also has a high correlation with starting time of the three objects and our heuristic only provides an estimated relationship between each other. In fact, as shown in red solid line of Fig. 7.5(b), such inferred relationship does not affect the final choice of the critical path.

[7]HAR Specification: http://www.softwareishard.com/blog/har-12-spec/. Downloading time period by HAR visualization is represented as different colors in the followings.

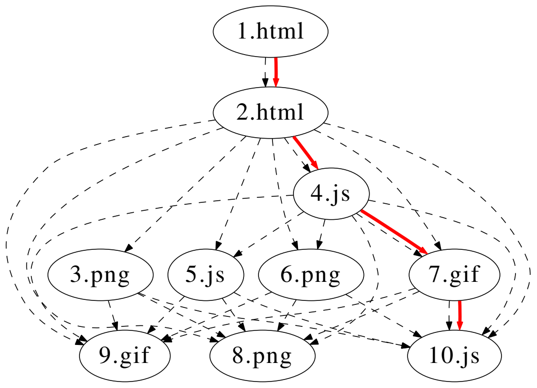
[8]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[9]http://en.wikipedia.org/wiki/Levenshtein_distance

- As we can also see that objects in the critical path generally take longer delays in the whole Google page. We will also show in the next sections, these objects can be also considered as key objects for the Google page.



(a) Timeline of Object Activity



(b) Extracted Dependency Graph with Critical Path

Figure 7.5: Critical Path Analysis Example for `Google` Page

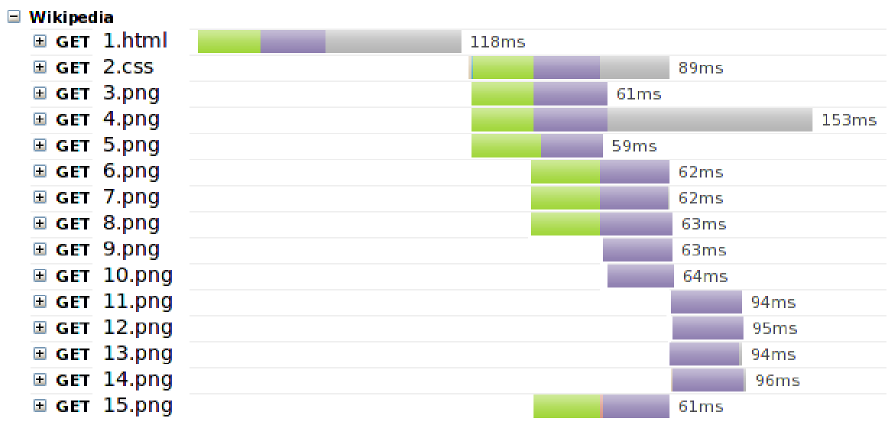
7.5.2.2 Wikipedia Page

We also show another example for `www.wikipedia.org` Web page. Download activity is shown in Fig. 7.6(a) and we see that there are more objects and parallelisms than previous case. After parsing the objects with the CPM model, we show the results in Fig. 7.6(b):

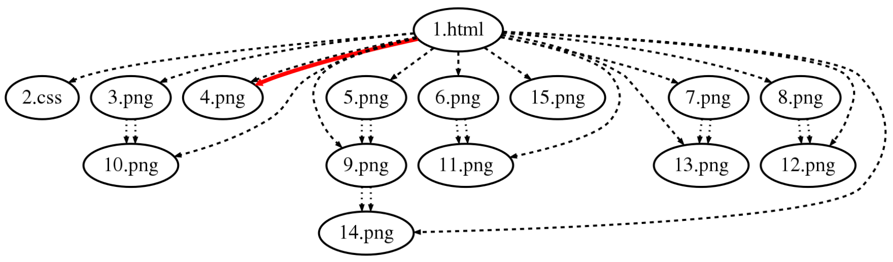
- The main object `1.html` is the root of the graph.
- All the other objects are referenced from the main HTML objects which are shown in dashed lines in Fig. 7.6(b).
- Besides the reference relationship, objects between each other can be also blocked due to maximum allowed parallel connections. We show this type of relationship with two parallel dotted lines in Fig. 7.6(b).
- From Fig. 7.6(a), the browser downloads multiple objects (e.g. `2.css`, `3.png` etc.) in parallel. Since object `5.png` completes slightly earlier than others, then the browser releases one channel to make `9.png` become active. Therefore we see this parental relationship from `5.png` to `9.png`

in Fig. 7.6(b). Similar dependency relationships are also shown in two parallel dotted lines in Fig. 7.6(b) such as link between `3.png` and `10.png` etc.

- Since object `4.png` takes longer than other downloadings, object `1.html` and `4.png` are selected in the critical path as shown in red solid line in Fig. 7.6(b).



(a) Timeline of Object Activity



(b) Extracted Dependency Graph with Critical Path

Figure 7.6: Critical Path Analysis Example for `wikipedia` Page

7.6 Usage of Critical Path Method

Based on the inherent nature of the critical path that *"activities in this path should have a direct impact on the whole project"*, in this section, we discuss usages of our method for the Web performance studies.

7.6.1 Performance Prediction

The first type of usage is to predict how performance will be changed under different conditions. We say it as the performance prediction under *What-if* scenarios.

7.6.1.1 *What-if* RTT Change

The first scenario we show is to predict how performance changes when the RTT increases. To illustrate this, we use two different types of Web pages `www.google.com`, `www.sigcomm.org` and perform similar browsing experiments as Sec. 7.5. These browsing samples are used as training sets to build the dependency graph. We then emulate RTT increasing using netem under three outgoing packet delay conditions: 0 ms, 100 ms and 200 ms. We use the page load time with 0 ms extra delay as baseline and predict its distribution for other cases by following steps:

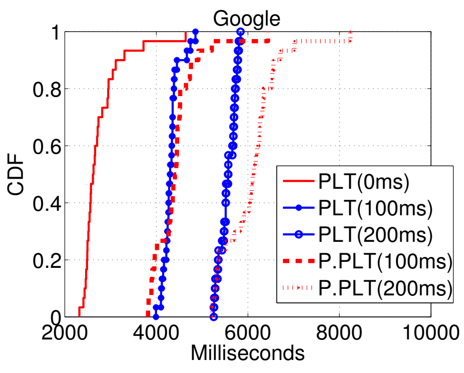
1) For each browsing in the baseline, we generate a dependency tree graph by the training sets;

2) Update each object delay in the dependency graph;

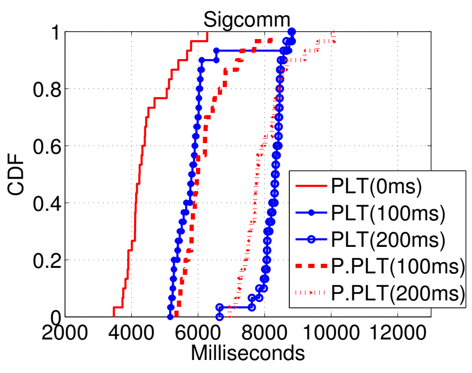
3) Find **new** critical path in the updated graph, and total delay in the new critical path is estimated as the page load time under new scenarios.

Among these steps, the second one about updating the downloading delay may be tricky. Since we rely on the coarse grain timing measurement at the browser level (e.g. as described in Sec. 7.5.1), we only estimate the increase in delay in a rough manner: whenever we measure DNS, TCP and HTTP query delays for an object, we update them by increasing the extra delays; For the data receiving delay, we should update the extra delay in each TCP round. From our measurement case, we estimate number of TCP rounds by using the receiving delay divided by median of TCP connecting delay for the same server host. While such estimation is in coarse-grained, as will be shown in the following, page load time can be still predicted within a small error margin for the different *what-if* scenarios. In this chapter, we compute the page load time as time interval between starting time of the first request and ending time of last bytes within a Web page.

Fig. 7.7 shows all the results for these two Web pages under different scenarios. For each of the extra delay settings, we browse the Web page 50 times with empty browser cache. Fig. 7.7(a) shows the CDF for `google.com` and Fig. 7.7(b) shows `sigcomm.org` Web pages. From these figures, we see that with each 100 ms increasing, both pages roughly have 1 seconds more for the page load time. The predicted results also keep closely in the distribution with the real page load times. We find that for 100 ms delay case, median relative errors are under 5%; while for 200 ms delay, both errors in the median case are under 10%. Even larger errors for the 90th percentile case, both errors are still under 15%.



(a) `www.google.com` Web Page



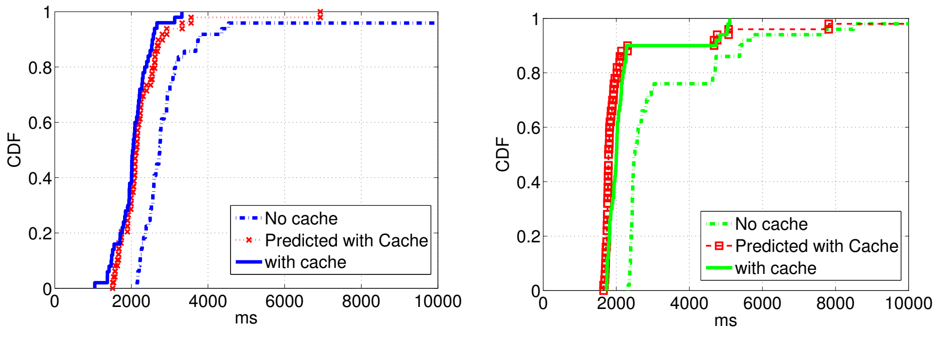
(b) `www.sigcomm.org` Web Page

Figure 7.7: How Page Load Time Changes when RTT Increases (PLT=Page Load Time, P.PLT=Predicted Page Load Time)

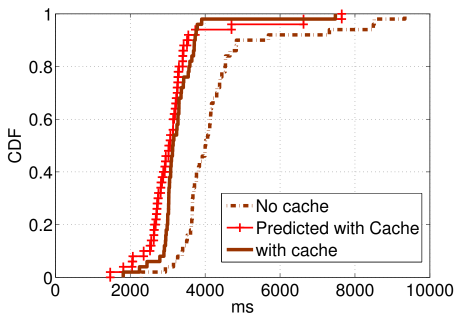
7.6.1.2 *What-if* Using Cache

Another example of using CPM is to predict how the download performance will be affected for the same page when a cache is used. In this scenario, when we find that one object can be completely cached by the browser, we only need to update its downloading delay to 0 ms in the original dependency graph and then recompute the new critical path.

We use three popular Web pages for different countries, namely `google`, `baidu` and `youtube`, as examples to show this scenario. For each page, we browse it back to back with the first time an empty cache while second time has the cache. After each couple of browsing, we clear the cache and then repeat the same procedure for 50 times. The experiments are done in a public wifi environment where network conditions vary a lot. We use browsing with empty cache as baseline and predict the page load time when using cache. To generate dependency graph in this case, we collaborate all the browsing samples with empty cache.



(a) `www.google.com` Web Page (b) `www.baidu.com` Web Page



(c) `www.youtube.com` Web Page

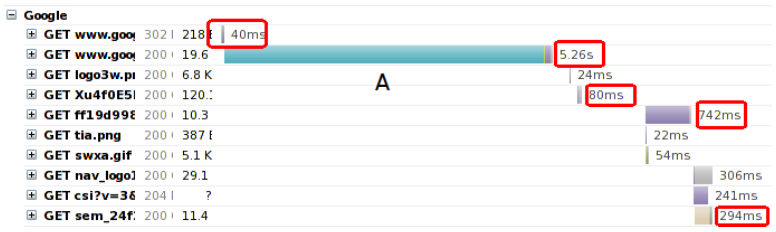
Figure 7.8: How Page Load Time Changes when Using Cache

Fig. 7.8 shows the predicted results for these three Web pages. We can see that the predicted results are pretty close to the real page load times when using cache. After computing the relative errors, we find the errors are less than 10% for most percentiles.

7.6.2 Performance Diagnosis

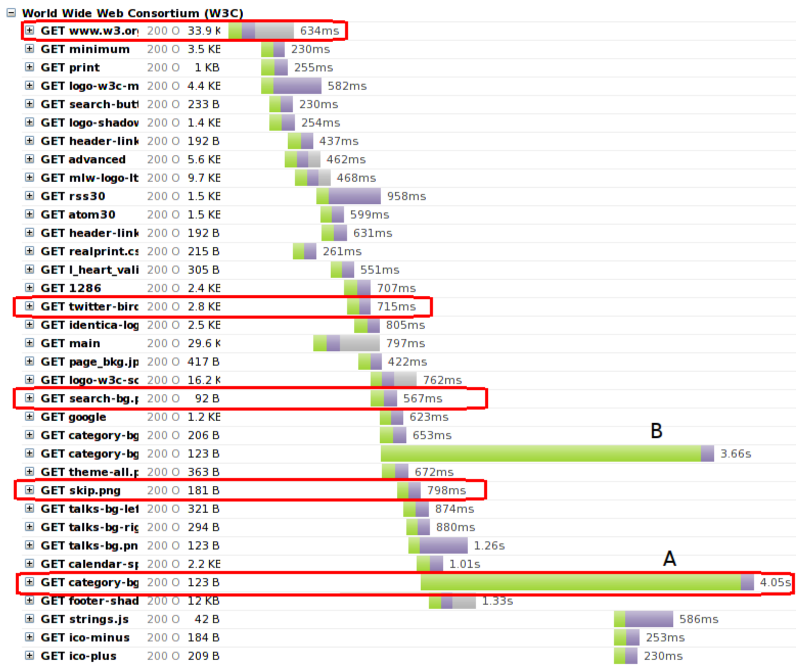
Besides performance prediction, we can also use the CPM for performance diagnosis. Since Web objects in the critical path have a direct impact on the final page load time, identifying anomalies in the download times for these objects can help us to discover performance bottlenecks.

Some examples are shown in Fig. 7.9: in Fig. 7.9(a), we show a `google` page browsing case where a long DNS delay occurs for the second object (object `A` in this figure). By adopting the CPM, we label the objects in the critical path with red squares. We can see that the the second object is included in the critical path. In Fig. 7.9(b), we show another example for `w3c` page. We see that two objects (object `A` and `B`) with abnormally high delays (in the TCP connecting phase) affect the total page load time. We also see that one of those objects object `A` is included in the critical path as well (labeled as red squares).



Google			
GET www.goo	302	218	40ms
GET www.goo	200	19.6	5.26s
GET logo3w.p	200	6.8 K	24ms
GET Xu4f0E5	200	120.	80ms
GET ff19d99	200	10.3	742ms
GET tia.png	200	387	22ms
GET swxa.gif	200	5.1 K	54ms
GET nav_logo	200	29.1	306ms
GET csi?v=3	204	?	241ms
GET sem_24f	200	11.4	294ms

(a) Poor `Google` Page Browsing with Load Time 7.8 seconds



World Wide Web Consortium (W3C)			
GET www.w3.or	200 O	33.9 K	634ms
GET minimum	200 O	3.5 KE	230ms
GET print	200 O	1 KB	255ms
GET logo-w3c-m	200 O	4.4 KE	582ms
GET search-butt	200 O	233 B	230ms
GET logo-shadov	200 O	1.4 KE	254ms
GET header-link	200 O	192 B	437ms
GET advanced	200 O	5.6 KE	462ms
GET mlw-logo-lt	200 O	9.7 KE	468ms
GET rss30	200 O	1.5 KE	958ms
GET atom30	200 O	1.5 KE	599ms
GET header-link	200 O	192 B	631ms
GET realprint.cs	200 O	215 B	261ms
GET I_heart_vali	200 O	305 B	551ms
GET 1286	200 O	2.4 KE	707ms
GET twitter-birc	200 O	2.8 KE	715ms
GET identica-log	200 O	2.5 KE	805ms
GET main	200 O	29.6 K	797ms
GET page_bkg.jp	200 O	417 B	422ms
GET logo-w3c-sc	200 O	16.2 K	762ms
GET search-bg.p	200 O	92 B	567ms
GET google	200 O	1.2 KE	623ms
GET category-bg	200 O	206 B	653ms
GET category-bg	200 O	123 B	3.66s
GET theme-all.p	200 O	363 B	672ms
GET skip.png	200 O	181 B	798ms
GET talks-bg-lef	200 O	321 B	874ms
GET talks-bg-rig	200 O	294 B	880ms
GET talks-bg.pn	200 O	123 B	1.26s
GET calendar-sp	200 O	2.2 KE	1.01s
GET category-bg	200 O	123 B	4.05s
GET footer-shad	200 O	12 KB	1.33s
GET strings.js	200 O	42 B	586ms
GET ico-minus	200 O	184 B	253ms
GET ico-plus	200 O	209 B	230ms

(b) Poor Page Browsing with Load Time 5.1 seconds on `www.w3.org`

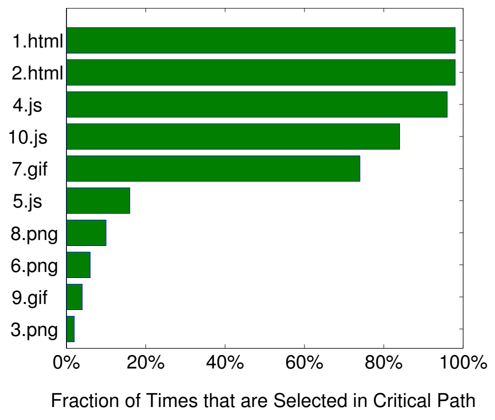
Figure 7.9: Performance Bottleneck Discovery by CPM

After identification of the performance bottlenecks, we can use *what-if* predictions to estimate how the performance can be improved when the browsing behavior experiences slight changes. For example, the `google` session as shown in Fig. 7.9(a) can be easily improved when DNS query for the second object is not lost. Since default for DNS retransmissions in Linux system is 5 seconds, page load time will be improved for around 5 seconds if there is no such DNS loss and the other objects experience the same performance. As for the `w3c` page shown in Fig. 7.9(b), since the TCP connecting delay affects mostly one single object (object `A`) in the critical path (e.g. slightly higher than 3 seconds), therefore we infer TCP handshake packet may get lost. If we assume there is no such loss at object `A` while other objects keep the same behaviors, by using our CPM, we predict the new page load time is still 4.7 seconds which means only $\frac{5.1-4.7}{5.1} = 8\%$ improvement to the original one. This is due to the fact that another object (object `B` in Fig. 7.9(b)) still experiences a high downloading delay.

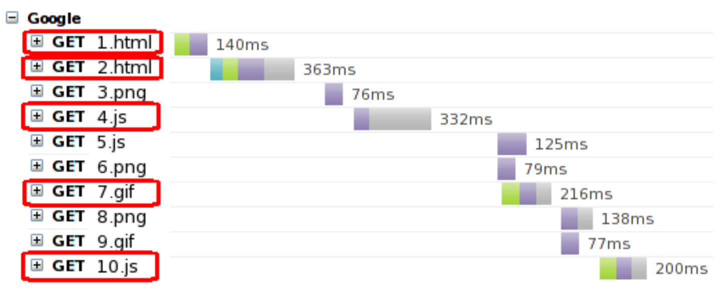
7.6.3 Key Object Extractor

Finally, we show another usage by our method, which is to extract key objects inside a page. A Web page may contain multiple objects, which may have different impact w.r.t. the whole Web page rendering time. We still use `google` experiment samples as used in Sec. 7.5.2.1. We extract one critical path for each web page download. Across all the downloads, we then count the number of times each object was part of the critical path. Fig. 7.10(a) shows the percentage of times.

We see that objects like the main HTML (`1.html` and `2.html`) and other script files (`4.js` and `10.js`) are counted into the critical path much more frequently (e.g. for >80% cases); however, icon objects like `3.png` are rarely critical. In this case, we say that objects that are frequently included in the critical path play more important roles for the `google.com` page rendering while others are not.



(a) Key Objects Distribution in `Google` Page



(b) Example of `Google` Page Object Activity

Figure 7.10: Key Object Extraction by CPM

7.7 Discussion

In this chapter, we presented a methodology called Critical Path Method to analyze the Web page download performance. To generate the critical path, extraction of the dependency graph is the key. We use some features by the Web browser itself and shared information of multiple browsing experiences to preform the dependency graph extraction. As we have shown, we browse a given Web page in an automatic way to accumulate enough samples. In this section, we discuss the impact of the number of shared samples on the quality of the dependency graph generation. We still use the Google case that are discussed in Sec. 7.5.2.1. We make the dependency graph generated by query all 50 shared samples as baseline; then we compare *similarity* between the baseline graph and other dependency graphs generated by query less amount of shared samples. Since the dependency graph is represented as a "tree", therefore, we quantify the similarity between two tree graphs G_{base} and G_1 by their *distances* computed as:

$$Dist(G_1, G_{base}) = Del(G_1, G_{base}) + Add(G_1, G_{base})$$

where $Del(G_1, G_{base})$ represents number of links that exist in graph G_1 but not exist in G_{base}, while $Add(G_1, G_{base})$ means number of links that do not exist in graph G_1 but exist in G_{base}. This distance formula represents total number of operations (either deleting or adding links) for G_1 to achieve the same graph of G_{base}. In our case, G_{base} refers to the baseline dependency graph and G_1 can be any dependency graph obtained using fewer samples.

Fig. 7.11 shows the results. For each test case, we randomly select a fixed number of samples from all the data. We generate 30 dependency graphs in each test and compute the average distance with the baseline graph. Due to the fact that after query other samples, we use correlation study (as discussed in Sec. 7.3.3) to check existence of some parental relationships, therefore, we also plot the results for different correlation threshold (β in Tab. 7.2) in Fig. 7.11. From this figure, we can see that distance between baseline and the dependency graphs inferred from small number of samples are typical larger. While increasing the shared sample size (e.g. from 5 to 20), distance decreases rapidly. When the shared sample size is large enough (e.g. >30), distance is stable around 1, meaning that the inferred dependency graph does not have much change comparing to the baseline graph. Moreover distance changes for different correlation thresholds are also similar.

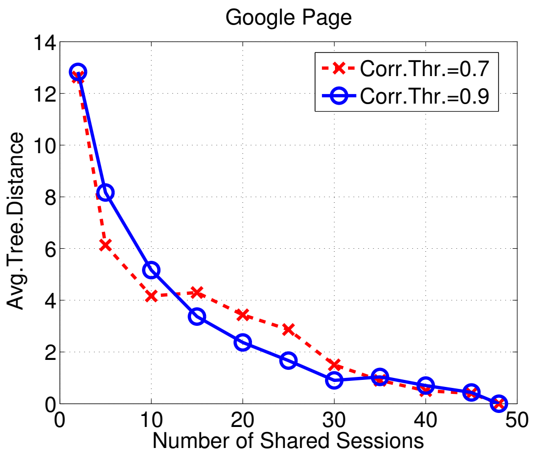


Figure 7.11: Discussions on Number of Shared Experiences (50 browsing samples in total)

7.8 Related Work

We can classify the literature related to Web performance studies as follows:

Characterization of Web traffic: Gebert et al. [56] show that HTTP flows dominate all the traffic in their 14 days measurement from an access network. Gehlen et al. [57] use one week worth of data to show that most of today's Web traffic is handled by a few organizations. Butkiewicz et al. [43] study Web page content-type features and their impact on the page performance. Ihm et al. [64] use long term log data to study the Web page content trend during recent five years.

Performance Modeling: Butkiewicz et al. [43] define a set of Web page complexity metrics and use linear regression to relate these metrics with page load time. Li et al. [73] focus on Google and Yahoo pages and propose WebProphet, a system for dependency extraction and page load performance prediction. Wischik [87] expresses page load time as a function of network conditions such as RTT and loss rate.

Performance Improvements: Some papers propose methods to improve the Web page load performance. For example, Google and Yahoo! publish recommendations [14] [34] on how to construct Web pages that load fast. Other papers study improvements to TCP that is used by HTTP to transfer the data. S. Radhakrishnan et al. [78] propose a fast open protocol for TCP connections that enables data exchange during the TCP handshake, which speeds up the Web page downloading. Other papers suggest to increase the TCP initial congestion window [51], to shorten the TCP retransmission timer [76].

Among these works, the closest to our work is the chapter on WebProphet [73] where authors propose to use controlled experiments to perturb the download time of the objects of a Web page to detect the object dependency relationships. As we have seen in the course of this chapter, such dependencies are also crucial in our our work. However, we use a completely different mechanism to extract this dependency information.

Conclusions for Part III

In this part, we have presented a new methodology called "Critical Path Method" for the Web performance study. The key idea of our methodology is to use some features of Web browser and shared information among different browsing experiences for the same page. Such shared experiences can be achieved either by other clients, or one's own browsing histories.

We first introduced some browser features that may cause inter-dependencies between each object. We used some examples to illustrate such phenomenons. After that, we formalized our intuition and provided the method to extract a critical path of a Web page download. We then used two Web pages `google.com` and `wikipedia.org`, as examples, to explain how the critical path would look like.

We finally proposed three usage scenarios based on our critical path method, named bottleneck discovery, performance prediction and also key object extraction. From our experiments, we illustrated promising perspectives of our methodology, for example, performance can be predicted with error less than 15% etc.

CHAPTER 8 Conclusion

"The true sign of intelligence is not knowledge but imagination."

Albert Einstein

8.1 Evaluation of the Thesis Work

In the introduction of this thesis, we have made several claims. We now revisit these claims and evaluate the overall thesis work.

The thesis claims that we have made are:

1. *Our initial platform which combines measurements at both higher user's level and lower network level can greatly reduce the gap between user's subjective QoE (Quality of Experience) and objectively measured QoS (Quality of Service).*

   Chapter 2 presented our initial platform. Our platform is browser-based, while different other components can be added such as packet capture etc.

   From the browser level measurements, we can easily obtain knowledge of end user perceived performances. To the best of our knowledge, we are the first that mainly use measurements at the browser level to study end user's experiences.

2. *Different kinds of clustering techniques can be adopted to study the Web performance*:

   - *Using traditional centralized clustering helps us to reveal the relationship between user level satisfaction (QoE) and lower network level (QoS) metrics.*
   - *Distributed clustering provides scalable approach to use the measurements performed of different users.*

   In Chapter 3, we proposed to use clustering to reveal the underlying relationship between QoE and QoS. We also show that clustering can be used to compare the performance between multiple homes.

Based on this finding, we extended the clustering method in Chapter 4 to work in a distributed way. We set up a multi-node prototype to perform the distributed clustering process among users in different home. Although each user only has a limited knowledge such as the centroids and counters, distributed clustering was able to provide interesting insights.

3. *We propose root cause analysis model based on passive measurements from user's wild browsing. Our method is implemented via the Mozilla development interfaces and stores the data in a database.*

   In Part II, we have presented our tool, FireLog, which is composed of client side measurements and server side diagnosis and analysis. We used experiments to show features of FireLog, such as low intrusiveness and high accuracy.

   We then introduced our diagnosis model for troubleshooting. Although there are multiple parameters to be set, we relied on some controlled experiments for their choices.

   Via a long-term (over five months) deployment at real home users, we have collected enough data for our diagnosis model. We found that, (i) there are always client side limitations across all the users; (ii) traditional metrics that evaluate network behaviors such as TCP round trip time can be sometimes insufficient to explain high page load time, since the performance can be limited by server side overhead.

4. *Sharing browsing experiences help us develop a novel approach in the analysis of the Web page browsing. Our novel approach named Critical Path Method can be useful for the Web performance analysis in several scenarios including providing what-if predictions and key object extraction.*

   In Part III, we have presented our methodology, named the "Critical Path Method", for the Web performance study. The key idea of our methodology is to use the shared information of different browsing experiences for the same Web page.

   We have introduced some browser features that may cause inter-dependencies between objects of a Web page. After that, we formalized our intuition and provided a method to extract such critical path of the objects that affect the whole page rendering. We then used two Web pages `google.com` and `wikipedia.org`, as examples, to explain in detail how the critical path would look like.

   We finally proposed three usage scenarios based on our critical path method, named bottleneck discovery, performance prediction and key object extraction. From our experiments, we have shown that our methodology is very promising and can, for example, predict download performance within 15% of error for all percentiles.

8.2 Future Work

There are several possible directions for the future work. We classify them into two categories: the first one is related to the tool development and second to model development.

Tool Development So far, in this thesis, we have studied the Web browsing performance using Firefox browser. It would be interesting to extend current works by:

- Developing more tools for other Web browsers: although Firefox is still one of the major Web browsers in the market, statistics [1] also show that the popularity of Google Chrome has increased at a high rate, from 10.8% in January 2010, up to 46.9% in December 2012. Therefore, porting our prototype into other browsers can be helpful in the following aspects:
    1) Increasing the measurement feasibility: This is also one of the main problems during our tool's deployment which prevents larger scale of measurements. Most of our contacted friends prefer using Google Chrome or Safari as their default Web browsers and only use Firefox for special needs.
    2) Cross-comparison between different browsers: In case we are able to collect data from multiple products, it will allow us compare their performances, not only for the same Web page, but also for the wild browsing.
- Porting our tool to the mobile users: our current development only focuses on the desktop users. However, analysis results show that the growth of the mobile users is much faster than desktop users, and will probably take over by the year 2015 [33]. At the same time, it is also known that Web access via mobile devices is still slow for most Web sites. Therefore, it would be also interesting to adopt our prototype to the mobile field.
- Exploring more browser-level measurement: due to the fact that much more interfaces are available for the browser development, it would useful to explore more information during Web page rendering and correlate them with lower level metrics. One example is to keep track of the page rendering progress. In case such progress is stuck, anomalies usually happen during that phase and user's experiences are likely degraded.

However, along with these possible future directions, there are also challenges. One of the most critical challenges is the maintenance: unlike lower-level measurement tools such as tcpdump or wireshark, browser-oriented tools have a common issue which is the compatibility problems with browser versions. For example, breakdown metrics used in the thesis can be only measured from Firefox 3.6, and socket information of a TCP connection is only available from Firefox 5.0. Mozilla also releases newer versions for Firefox quite frequently[1] in recent periods, which can potentially make our tools incompatible with newer versions.

Model Development Another direction for the future work may relate to more intelligent diagnosis models. Although we have proposed a diagnosis model in this thesis (Chapter. 6), there are still limitations. For example, a fully automatic collaborative model can be developed as the improvement for our current stage. Garcia et al. [55] propose a collaborative system for network troubleshooting based on multi-agents and Bayesian network; similar architectures are also proposed by the DYSWIS system [70] and KOWLAN [54].

As we have also seen in the thesis, correlations exist between large delays and user dis-satisfaction. However, *correlation does not necessarily imply causality*. There are, for instance, many reasons for experiencing large delays such as large buffer at the access link or an overloaded proxy between the client and server. It would be interesting to use the framework developed by J. Pearl [66] to study causal relationships for the causality reasoning and inferences.

[1]In 2012, Firefox updates its version almost every month [24].

Bibliography

[1] Browser Statistics and Trends. http://www.w3schools.com/browsers/browsers_stats.asp.

[2] Chrome Developer Tools. https://developers.google.com/chrome-developer-tools/docs/timeline.

[3] Connectivity Enhancements in Windows Internet Explorer 8. http://msdn.microsoft.com/en-us/library/cc304129(VS.85).aspx.

[4] Cosine Similarity. http://en.wikipedia.org/wiki/Cosine_similarity.

[5] Critical Path Method. http://en.wikipedia.org/wiki/Critical_path_method.

[6] Distinctive Features Of SQLite. http://www.sqlite.org/different.html.

[7] Dynatrace. http://ajax.dynatrace.com/.

[8] Ellacoya Networks News. Web Traffic Overtakes Peer-to-Peer (P2P). http://www.circleid.com/posts/web_traffic_overtakes_p2p_bandwidth.

[9] Firebug. http://getfirebug.com/.

[10] For Impatient Web Users, an Eye Blink Is Just Too Long to Wait. http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html.

[11] Gomez White Paper: Why Web Performance Matters: Is Your Site Driving Customers Away? http://www.gomez.com/pdfs/wp_why_web_performance_matters.pdf.

[12] Google Project: Make the web faster. https://developers.google.com/speed.

[13] Google mod_pagespeed Project. https://developers.google.com/speed/pagespeed/mod.

[14] Google Tutorials for Fast Web Page. https://developers.google.com/speed/articles.

[15] Google web toolkit: Speed tracer. https://developers.google.com/web-toolkit/speedtracer/.

[16] Gossip protocol. http://en.wikipedia.org/wiki/Gossip_protocol.

[17] HTTP Achieve Specification. http://www.softwareishard.com/blog/har-12-spec/.

[18] IE Developer Tools. http://msdn.microsoft.com/en-us/library/gg130952.

[19] Initcwnd Settings of Major CDN Providers. http://www.cdnplanet.com/blog/initcwnd-settings-major-cdn-providers/.

[20] Internet End-to-end Performance Monitoring. http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl.

[21] List of HTTP header fields. http://en.wikipedia.org/wiki/List_of_HTTP_header_fields.

[22] Maxmind. http://www.maxmind.com.

[23] Mozilla. https://developer.mozilla.org/en/.

[24] Mozilla Wiki. https://wiki.mozilla.org/Releases.

[25] Mozilla XPCOM API Reference. https://developer.mozilla.org/en-US/docs/XPCOM_API_Reference.

[26] Postgresql database. http://www.postgresql.org/.

[27] SPDY Project. http://dev.chromium.org/spdy/spdy-whitepaper.

[28] TCP Fast Open in Linux 3.6. http://lwn.net/Articles/508865/.

[29] TCP Fast Open in Linux 3.7. http://www.zdnet.com/linux-3-7-arrives-arm-developers-rejoice-7000008638/.

[30] The WorldWideWeb Browser. http://www.w3.org/People/Berners-Lee/WorldWideWeb.html.

[31] WebPagetest. http://www.webpagetest.org/.

[32] Website of the world's first-ever web server. http://info.cern.ch.

[33] White Paper of Internet Trends 2010. http://zapd.me/directory/research-data/Whitepaper-Internet-Trends-apr2010.pdf.

[34] Yahoo! Speeding Up. http://developer.yahoo.com/performance/rules.html.

[35] S. Agarwal, N. Liogkas, P. Mohan, and V.N. Padmanabhan. WebProfiler: Cooperative Diagnosis of Web Failures. In *Proceedings of the 2nd international conference on COMmunication systems and NETworks*, pages 288–298, January 2010.

[36] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS Resolvers in the Wild. In *Proceedings of IMC'10*, Melbourne, Australia, November 2010.

[37] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. Padmanabhan, and G. M. Voelker. NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, April 2009.

[38] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. *IETF RFC3390*, 2002.

[39] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. *IETF RFC5681*, 2009.

[40] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *IETF RFC2581*, 1999.

[41] S. Balzano. Master Thesis: A Distributed Architecture for Web Browsing Troubleshooting. http://www.eurecom.fr/~balzano/msthesis/thesis.unina.SB.pdf.

[42] J. Brutlag. Speed Matters for Google Web Search. Technical report, Google Inc., June 2009.

[43] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proceedings of IMC'11*, Berlin, Germany, November 2011.

[44] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou. Instrumenting Home Networks. In *HomeNets, ACM SIGCOMM Workshop on Home Networks*, New Delhi, India, September 2010.

[45] K.T. Chen, C.C. Tu, and W.C. Xiao. OneClick: A Framework for Measuring Network Quality of Experience. In *Proceedings of IEEE INFOCOM 2009*, pages 702–710, Rio de Janeiro, Brazil, April 2009.

[46] Y. Chen. Dynamic Distributed K-means Clustering System. Technical report, Eurecom. `http://www.eurecom.fr/~cui/techrep/DDkmeansChen.pdf`.

[47] Y. Chen, S. Jain, V. K. Adhikari, and Z. Zhang. Characterizing Roles of Front-end Servers in End-to-End Performance of Dynamic Content Distribution. In *Proceedings of IMC'11*, Berlin, Germany, November 2011.

[48] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. *IETF. I-D draft-ietf-tcpm-fastopen-02*, 2012.

[49] S. Datta, C. Giannella, and H. Kargupta. K-Means Clustering Over a Large, Dynamic Network. In *Proceedings of SIAM Int'l Conf. Data Mining*, Maryland, USA, April 2006.

[50] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A Browser-based Network Measurement Platform. In *Proceedings of IMC'12*, Boston, MA, USA, November 2012.

[51] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An Argument for Increasing TCP's Initial Congestion Window. *ACM Computer Communication Review*, 40:27–33, 2010.

[52] A. Finamore, M. Mellia, and M. Meo. Mining unclassified traffic using automatic clustering techniques. In *TMA 2011, third International Workshop on Traffic Monitoring and Analysis*, Vienna, Austria, April 2011.

[53] D. F. Galletta, R. Henry, S. McCoy, and P. Polak. Web Site Delays: How Tolerant are Users? *Association for Information Systems*, 5:1–28, 2004.

[54] S. Garcia-Gomez, J. Gonzalez-Ordas, F. J. Garcia-Algarra, R. Toribio-Sardon, A. Sedano-Frade, and F. Buisan-Garcia. KOWLAN: A Multi Agent System for Bayesian Diagnosis in Telecommunication Networks. In *Proceedings of IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 195–198, Milan, Italy, September 2009.

[55] F. J. García-Algarra, P. Arozarena-Llopis, S. García-Gómez, and Á. Carrera-Barroso. A Lightweight Approach to Distributed Network Diagnosis under Uncertainty. In *Intelligent Networking and Collaborative Systems*, pages 74–80, Barcelona, Spain, November 2009.

[56] S. Gebert, R. Pries, D. Schlosser, and K. Heck. Internet Access Traffic Measurement and Analysis. In *TMA 2012*, Vienna, Austria, 2012.

[57] V. Gehlen, A. Finamore, M. Mellia, and M. Munafo. Uncovering the Big Players of the Web. In *TMA 2012*, Vienna, Austria, 2012.

[58] A. Hafsaoui. *Application Level Performance in Wired and Wireless Environments*. PhD thesis, TELECOM ParisTech, 2011.

[59] A. Hafsaoui, G. Urvoy-Keller, D. Collange, M. Siekkinen, and T. En-Najjary. Understanding the impact of the access technology: the case of web search services. In *TMA 2011, third International Workshop on Traffic Monitoring and Analysis*, Vienna, Austria, April 2011.

[60] F. Hernandez-Campos, K. Jeffay, and F. D. Smith. Tracking the Evolution of Web Traffic: 1995-2003. In *Proceedings of MASCOTS*, October 2003.

[61] T. Hossfeld, S. Biedermann, R. Schatz, A. Platzer, S. Egger, and M. Fiedler. The Memory Effect and Its Implications on Web QoE Modeling. In *Proceedings of 23rd International Teletraffic Congress*, San Francisco, USA, 2011.

[62] J.A. Hoxmeier and C. DiCesare. System response time and user satisfaction: An experimental study of browser-based applications. In *Proceedings of the Association of Information Systems Americas Conference*, Long Beach, California, USA, 2000.

[63] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and V. Bahl. Anatomizing Application Performance Differences on Smartphones. In *International Conference On Mobile Systems*, 2010.

[64] S. Ihm and V. S. Pai. Towards Understanding Modern Web Traffic. In *Proceedings of IMC'11*, Berlin, Germany, November 2011.

[65] ITU-T. Estimating End-to-End Performance in IP Networks for Data Applications. *ITU-T Recommandation G.1030*, November 2005.

[66] J. Pearl. *Causality Models, Reasoning and Inference 2nd Edition*. Cambridge University Press, 2009.

[67] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. HostView: Annotating End-host Performance Measurements with User Feedback. In *HotMetrics, ACM Sigmetrics Workshop*, New York, NY, USA, June 2010.

[68] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. Performance of Networked Applications: The Challenges in Capturing the User's Perception. In *ACM SIGCOMM Workshop on Measurements Up the Stack*, Toronto, Canada, August 2011.

[69] T. Karagiannis, E. Athanasopoulos, C. Gkantsidis, and P. Key. HomeMaestro: Order from Chaos in Home Networks. Technical Report MSR-TR-2008-84, Microsoft Research, May 2008.

[70] K.-H. Kim, V.l Singh, and H. G. Schulzrinne. DYSWIS: Collaborative Network Fault Diagnosis - Of End-users, By End-users, For End-users. Technical report, Columbia University, 2011.

[71] R. Kohavi and R. Longbotham. Online experiments: Lessons learned. *IEEE Computer*, 40(9):103–105, 2007.

[72] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating The Edge Network. In *Proceedings of IMC '10*, pages 246–259, New York, NY, USA, 2010. ACM.

[73] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y. Wang. WebProphet: Automating Performance Prediction for Web Services. In *Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation*, San Jose, US, April 2010.

[74] J. S. Miller, A. Mondal, R. Potharaju, P. A. Dinda, and A. Kuzmanovic. Understanding end-user perception of network problems. In *ACM SIGCOMM Workshop on Measurements Up the Stack*, Toronto, Canada, August 2011.

[75] F. Nah. A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait? In *Proceedings of AMCIS*, 2003.

[76] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. *IETF RFC6298*, 2011.

[77] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, USA, 2000. Morgan Kaufmann.

[78] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP Fast Open. In *Proceedings of CoNEXT'11*, Tokyo, Japan, December 2011.

[79] A. Reggani, F. Schneider, and R. Teixeira. An End-Host View on Local Traffic at Home and Work . In *Proceedings of PAM'12*, Vienna, Austria, 2012.

[80] S. Souders. *High Performance Web Sites: Essential Knowledge for Front-End Engineers*. O'Reilly Media, 2007.

[81] S. Souders. *Even Faster Web Sites*. O'Reilly Media, 2009.

[82] R. Schatz, S. Egger, and K. Masuch. The Impact of User Fatigue and Test Duration on the Reliability of Subjective Quality Ratings. *Journal of the Audio Engineering Society*, 59(10), March 2012.

[83] F. Schneider, B. Ager, G. Maier, A. Feldmann, and S. Uhlig. Pitfalls in HTTP Traffic Measurements and Analysis. In *PAM'12*, Vienna, Austria, 2012.

[84] M. Siekkinen. *Root Cause Analysis of TCP Throughput: Methodology, Techniques, and Applications*. PhD thesis, University of Nice-Sophia Antipolis, 2006.

[85] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and D. Collange. A Root Cause Analysis Toolkit for TCP. *Computer Networks*, 52(9):1846–1858, 2008.

[86] M. Allman V. Paxson. Computing TCP's Retransmission Timer. *IETF RFC2988*, 2000.

[87] D. Wischik. Short messages. In *Royal Society workshop on networks: modelling and control*, September 2007.

APPENDIX A Résumé en Français

A.1 Introduction

A.1.1 Motivation

Le World Wide Web (**WWW** ou **Web**) est un système global de documents hypertextes rendus par un navigateur Web. Il a été inventé par Tim Berners-Lee en 1989 au CERN. L'année d'après, il a également construit le premier navigateur Web [30] et le serveur Web [32].

Depuis ces années, les technologies construites à partir de celui-ci se développent à un rythme effréné. Un rapport [60] déclare que entre 1995 à 2003, le trafic Web constituait la plus grande partie du trafic Internet. Des observations similaire apparaissent également dans [8], qui observe que le trafic Web a déjà dépassé, le trafic Peer-to-Peer (P2P) et consommé le plus grand pourcentage de la bande passante sur le réseau. Pour l'utilisateur final, les applications liées au Web, par exemple la consultation d'un article Wikipedia, l'accès à une page d'information, les achats en ligne, la visualisation le contenu généré par l'utilisateur tel que sur YouTube ou sur Dailymotion, deviennent également l'utilisation la plus populaire pour le surf sur Internet [79].

De nombreux travaux ont montré que la mauvaise performance liée au Web peut avoir d'importants impacts à la fois pour les entreprises et les utilisateurs finaux. Google, en tant que fournisseur de services Web le plus populaire, fait de grands efforts pour améliorer ses performances [12]. Une de leur expérience [42] montre clairement que si la latence augmentait de 100 à 400 ms, le nombre quotidien de recherches par utilisateur serait réduit de 0.2% à 0.6%. Un rapport de Gomez [11] apporte des conclusions similaires: pour un site Web de e-commerce d'une taille de $100,000/jour, 1 seconde supplémentaire de délai signifie une réduction du nombre de clients de 2% et la réduction de 2.5 millions de dollars dans le chiffre d'affaire annuel. `Amazon.com` [71] estime également que chaque augmentation de 100 ms dans le temps de chargement peut diminuer leurs ventes de 1%.

D'autre part, un utilisateur final peut facilement devenir impatient et abandonner la page actuelle lorsque celle-ci devient trop lente (par exemple [11] [53] [62] [75]), voire même de passer sur le site web d'un concurrent ([10] [43]).

Sur base de ces considérations, nous avons besoin de systèmes de diagnostic ou des méthodologies afin d'analyser la performance Web pour l'utilisateur final. Ces outils peuvent aider à identifier les goulets d'étranglement de performances pour le pages Web consultées. Pour les fournisseurs de services, il peut être également utile de quantifier la qualité de l'expérience (QoE) pour leurs propres clients. Nous nous concentrons, dans cette thèse, sur le développement d'outils ou de méthodes de diagnostic des performance Web du point de vue de l'utilisateur.

A.1.2 Objectifs de la Thèse

Pour cette thèse, nous formulons les objectifs suivantes:

I Notre plate-forme initiale, qui combine des mesures au niveau de l'utilisateur et au niveau du réseau en lui-même, peut réduire considérablement l'écart entre la QoE *subjective* de l'utilisateur (Quality of Experience) et la mesurée QoS (Quality of Service) *objectivement*.

II Plusieurs types de techniques de clustering peuvent être adoptées pour étudier les performances Web:

- Le clustering classique centralisé permet de souligner la relation entre la satisfaction de l'utilfisateur (QoE) et au niveau inférieur des métriques réseau (QoS).
- Le clustering distribué fournit une approche évolutive à l'utilisation de mesures effectuées sur différents utilisateurs.

III Nous proposons des modèles d'analyse des causes principales basés sur des mesures passives de navigation d'utilisateurs. Notre méthode est basée sur les interfaces de développement de Mozilla et enregistre les données dans une base de données.

IV Le partage des expériences de navigation nous permet de développer une nouvelle approche dans l'analyse de la navigation page Web. Notre nouvelle approche, nommée méthode du chemin critique, peut être utile pour l'analyse des performances Web dans plusieurs scénarios, y compris la prévision "what-if" et l'extraction de l'objet clé.

A.1.3 Organisation de la Thèse

Cette thèse est divisée en trois parties:

Dans la première partie, nous présentons l'architecture initiale (chapitre 2) pour évaluer les performances Web ainsi que notre méthodologie d'analyse basée sur un navigateur. Sur la base de ce premier prototype, nous utilisons des techniques de clustering, à la fois centralisée (chapitre 3) et distribuée (chapitre 4) de manière à analyser et diagnostiquer une mauvaise exécution de la navigation Web. Ceci adresse les objectifs I et II du paragraphe précédent.

La deuxième partie, aborde le troisième objectif III de la thèse. Nous décrivons d'abord l'adaptation de notre outil au déploiements à domicile (chapitre 5). Ensuite, nous discutons des scénarios de limitation de performance. Enfin, nous développons un modèle automatisé pour le diagnostic de performance (chapitre 6).

Dans la troisième partie, nous proposons une nouvelle méthodologie pour analyser le Web. Basée sur les des discussions des chapitres précédents précédents nous considérons tous les objets d'une page Web comme une seule entité. Dans ce cas, nous nous concentrons sur la recherche d'événements critiques lors du rendu d'une page.

Finalement, le chapitre 8 conclus la thèse, nous concluons la thèse et propose plusieurs perspectives d'avenir.

A.2 Contributions Principales

Dans cette section, nous présentons quelques résultats clés décrits dans la thèse.

A.2.1 Analyse de la performance Web basée sur les expériences de l'utilisateur

Dans la première partie de la thèse, nous décrivons et justifions notre méthodologie d'analyse des performances de la navigation Web, en particulier la performance perçue par l'utilisateur final.

Measurement Set-up Nous présentons d'abord notre configuration initiale, qui est schématisée dans la Fig. A.1, pour la mesure de la page navigation Web. Nous décrivons son fonctionnement à différents niveaux et comment nous combinons les mesures. Comme nous pouvons le voir sur cette figure, nous utilisons un plugin du navigateur pour mesurer les événements clés comme le temps de chargement et de rendu du une page web. Dans cette thèse, nous utilisons le "temps de chargement" à titre indicatif pour la perception des performances par l'utilisateur final lors de sa navigation. Nous utilisons également la traditionnelle capture de paquets pour enregistrer les données brutes lors de la navigation sur une page Web. Dans ce cas, nous pouvons facilement calculer les métriques de bas niveau en fonction de nos besoins. Enfin, nous chargeons toutes les données collectées dans une base de données pour le post-traitement.

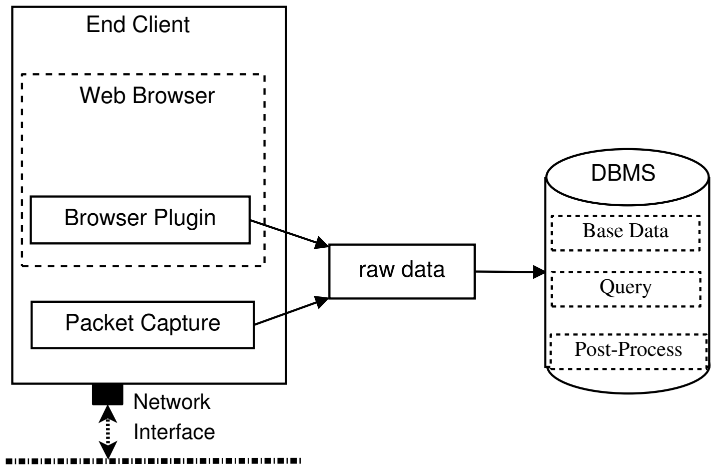


Figure A.1: Initial Platform Architecture

Exemple d'Utilisation Pour illustrer l'utilisation de cette environnement de mesure, nous déployons notre système aux domiciles de trois utilisateurs différents. Nous demandons à ces utilisateurs de naviguer sur des pages Web au hasard pendant deux jours, S'ils ne sont pas satisfait du temps de chargement d'une page, ils peuvent cliquer sur un bouton intégré au navigateur par le plugin. Grâce à cette feedback des utilisateurs, nous catégorisons une page Web comme "bonne" si il n'y a aucun événement “clic”, sinon comme “mauvaise”. On définit la liste de l’indice de performance clé (KPI) des métriques mesurée au niveau des paquets et les utilisons en tant que représentant d’une page Web. Nous prenons toutes les sessions de navigation Web et utilisons kmeans pour les regrouper entre eux au niveau mondial. La Fig. A.2(a) montre les indicateurs de performance pour chaque groupe et la Fig. A.2(b) montre les temps de chargement des pages chaque groupe. Le Tab. A.1 répertorie les valeurs médianes pour les différents paramètres et le pourcentage de sessions de bonne et mauvaise qualité.

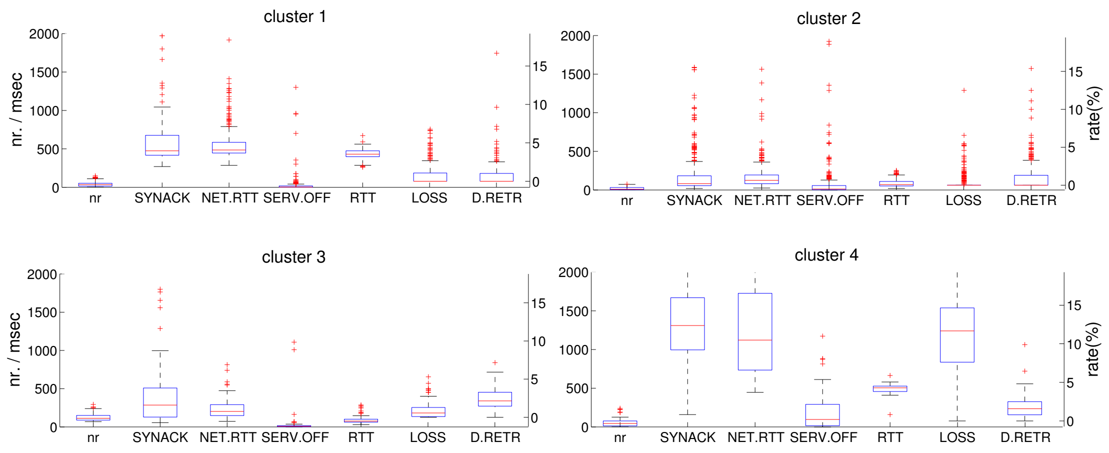
- Les pages Web du groupe 2 ont des temps de chargement généralement courts, qui est, pour plus de 80% des séances, en dessous de 10 secondes. Les indicateurs KPI liés aux retards et pertes sont tous très bas. Dans seulement 6% des cas, les utilisateurs ont été insatisfaits avec le temps qu’il a fallu pour charger la page, ce qui est le taux le plus bas de tous les groupes.
- À l’opposé, les utilisateurs ont été insatisfaits dans 64% des cas de pages Web du groupe 4. Ce n’est pas surprenant, puisque 90% de ces sessions Web ont besoin de plus de 10 secondes pour se charger complètement, et 50% des sessions ont même besoin de plus de 100 secondes. Les indicateurs KPI liés aux retards et pertes sont très élevés: le délai médian du TCP Handshake et la requête HTTP sont de plus de 1 seconde et le taux de perte moyen est supérieur à 10%. Après avoir analysé en détails les sessions Web de ce groupe, on trouve que la plupart d’entre elles accèdent au même domaine.
- L’accès aux pages Web dans les groupes 1 et 3 ont des temps de chargement similaires et sont classés comme pauvres 21% et 42% du temps, respectivement. Cependant, les caractéristiques des pages Web et les indicateurs de performance clés pour ces deux groupes sont différents: les pages Web en groupe 3 ont normalement plus d’objets que celles du groupe 1. Le Tab. A.1 montre aussi que le RTT médian du groupe 3 est d’environ 70 ms tandis que pour le groupe 1 il est normalement d’environ 400 ms. La valeur médiane de la perte de paquets et le taux de retransmission dans le groupe 3 sont un peu plus élevés que dans le groupe 1.

Table A.1: Statistics of the **4** Clusters (*TOT* and *poor%* refer as total number of Web sessions and the percentage of poor sessions in that cluster, respectively).

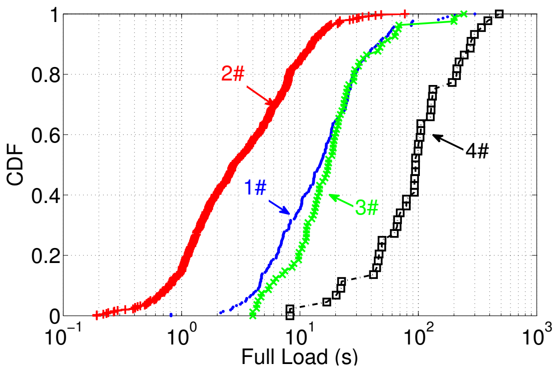
	Median values								
clusterID	*nr.*	*SYNACK*	*NET.RTT*	*SERV.OFF*	*RTT*	*LOSS*	*D.RETR*	*TOT*	*poor%*
Cluster 1	34	474	485	3	**431**	0.0%	0.0%	209	21%
Cluster 2	10	82	124	12	70	0.0%	0.0%	399	**6%**
Cluster 3	**113**	285	202	8	**76**	0.6%	2.1%	81	42%
Cluster 4	44	**1310**	**1122**	96	**505**	**11.7%**	1.6%	44	**64%**

Avec cet exemple, nous montrons le scénario d’utilisation utilisation de notre dispositif de mesure et la technique de clustering pour étudier la relation entre QoE (Quality of Experience) et QoS (Quality of Service). Nous montrons que le temps de chargement et la perception subjective de l’utilisateur peuvent en quelque sorte être compatibles avec les paramètres mesurés KPI.

Conclusion de la partie I Dans la première partie de la thèse, nous avons d’abord présenté la plate-forme de mesure basique pour nos études. Le navigateur Web étant le composant le plus proche l’utilisateur final pour surfer sur le Web de, notre plate-forme est basée sur un navigateur, mais différents autres composants peuvent être ajoutés tels que la capture de paquets, etc. Grace aux mesures depuis le navigateur, nous pouvons facilement acquérir l’expérience des performances perçues par l’utilisateur final.



(a) Wild Browsing Clustering Results for KPI Metrics



(b) Page Load Time in Clusters

Figure A.2: Clustering of All User Sessions in 4 Clusters

Nous avons alors proposé d'utiliser le clustering pour révéler la relation intrinsèque entre QoE et QoS. Nous avons montré que la perception de l'utilisateur, manifestée en un sentiment subjectif peut en quelque sorte être compatible avec les paramètres objectif mesurés. Nous avons également montré que ce clustering peut être utilisé comme méthode de collaboration entre plusieurs maisons.

Partant de ce constat, nous avons rendu le processus de clustering distribué entre les domiciles des différents utilisateurs. Bien que chaque utilisateur n'ait qu'une connaissance limitée comme par exemple les centres de gravité et les compteurs, nous avons pu réussi à souligner plusieurs faits intéressantes grâce à des comparaisons entre les différentes maisons comme des problèmes de réseau local ou des problèmes de sélection de serveur DNS par défaut.

A.2.2 Dépannage des sessions Web

Dans la partie précédente de la thèse, nous avons utilisé l'analyse humaine pour étudier les performances Web. Nous avons particulièrement expliqué les pages Web avec des performances médiocres. Dans la deuxième partie de la thèse, nous automatisons la méthode de diagnostic, et proposons notre méthode pour diagnostiquer les mauvaises performances de navigation.

Scénario Un grand nombre de mesures doivent être faites pour rendre une page Web dans son entièreté, et un certain nombre de composants est impliqué dans la génération de la transmission et le rendu du contenu. Comme le montre la Fig. A.3, ces composants sont: (i) l'ordinateur du client, (ii) la boucle locale, (iii) la partie restante de l'Internet, et (iv) les serveurs. Le ralentissement d'un de ces éléments de ces éléments aura une incidence sur le temps de chargement. L'objectif de cette partie est d'identifier quel composant est le principal responsable de la lenteur de chargement de la page Web.

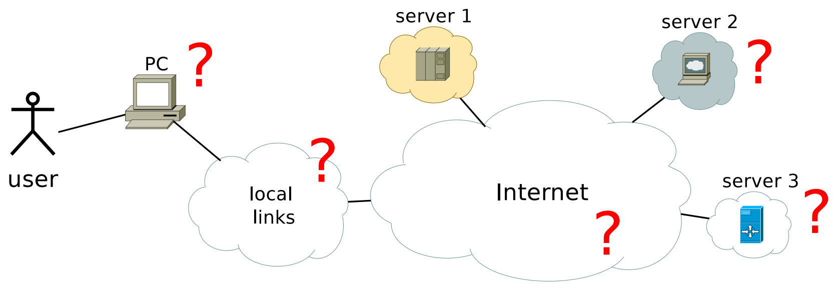


Figure A.3: End to end path

Développement d'Outil L'outil pour notre diagnostic à partir duquel FireLog et se compose de deux parties: le moteur côté client, pour la mesure, et le référentiel du serveur pour analyse. Le côté client est un plugin du navigateur Firefox à partir duquel toutes les mesures sont effectuées. Du côté serveur, toutes les données de mesure des clients sont acceptées, puis transférées dans une base de données pour le diagnostic.

Les navigateurs modernes offrent un ensemble riche d'événements utiles pour notre mesure, et notre plugin a été développé pour être en mesure de détecter de tels événements qui sont utilisés pour calculer les paramètres de notre intérêt. La Fig. A.4 illustre la mesure et aussi les indicateurs clés pour le diagnostic.

Du côté du serveur, nous acceptons les données transférées du client et les chargeons dans la base de données. Selon les paramètres mesurés, nous calculons plusieurs dizaines de facteurs pour quantifier les scénarios de goulot de performance. Nous définissons notre règle de diagnostic comme illustré à la Fig. A.5. Nous pouvons voir que l'idée générale est un arbre de décision où des seuils différents sont utilisés à chaque étape de décision.

Expérience de laboratoire Pour calibrer les seuils pour le diagnostic, nous effectuons plusieurs expériences dans un environnement de laboratoire contrôlé. Nous montrons un exemple de seuil de limitation du côté client (th_c dans la Fig. A.5) dans la Fig. A.6(a). Nous parcourons la page principale de Yahoo! tout en affectant le maximum de CPU de l'ordinateur client à Firefox, et observons les scores de limitation calculés. Nous voyons que lorsque le maximum autorisé l'utilisation du processeur par Firefox est compris entre 30% et 100% (pas de limite), ces valeurs de ces scores sont proches de 0, tandis que pour 20% avec une limite de CPU de 10%, les valeurs de score sont divisées. Par exemple pour le cas médian, une allocation de CPU de 20% a un score aux alentours de 0.2, ce qui signifie que pendant 20% du temps passé à telecharger l'objet est inactif. D'autres pages testées

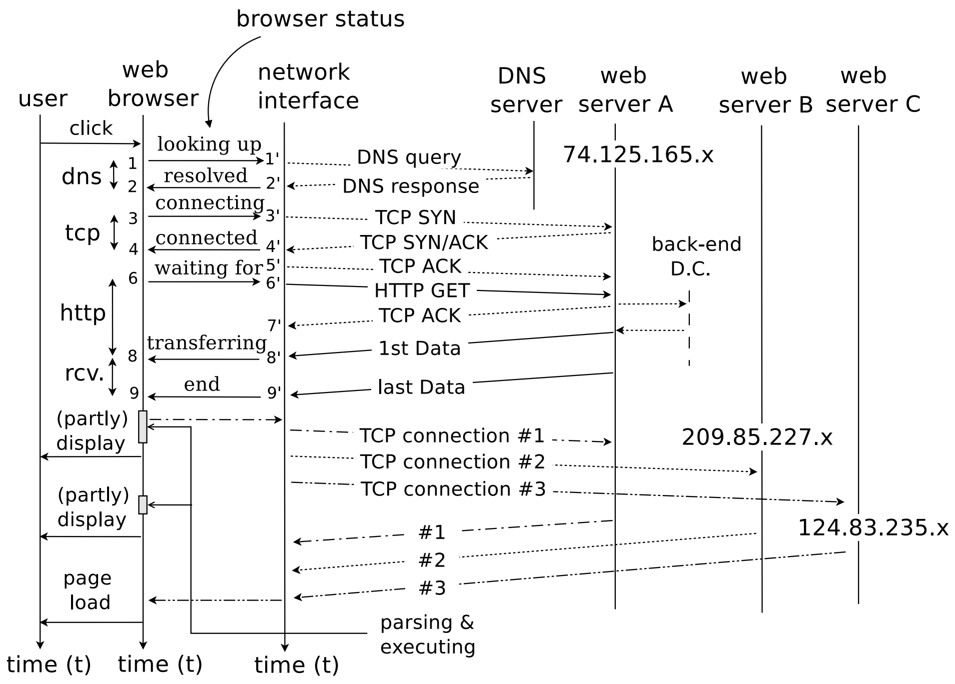


Figure A.4: Main Metrics Used by FireLog.

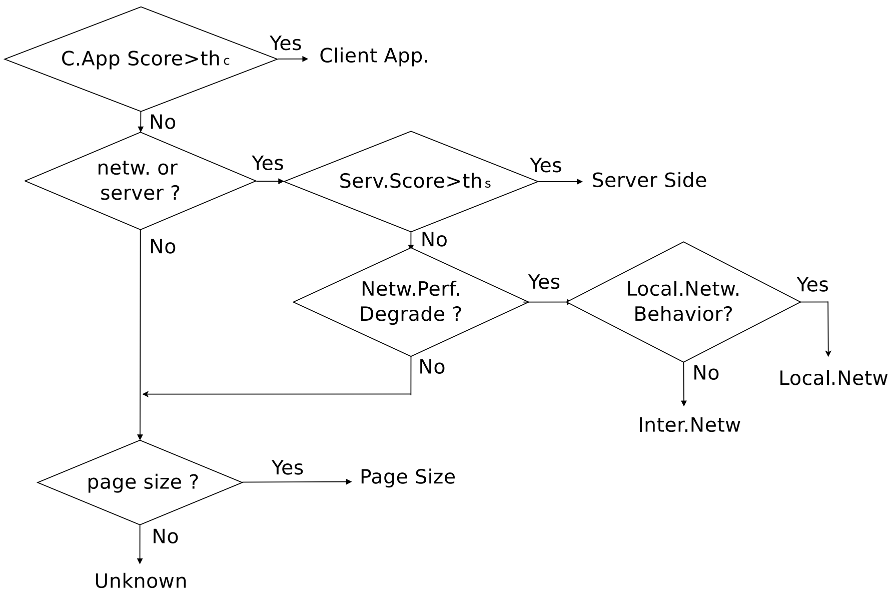
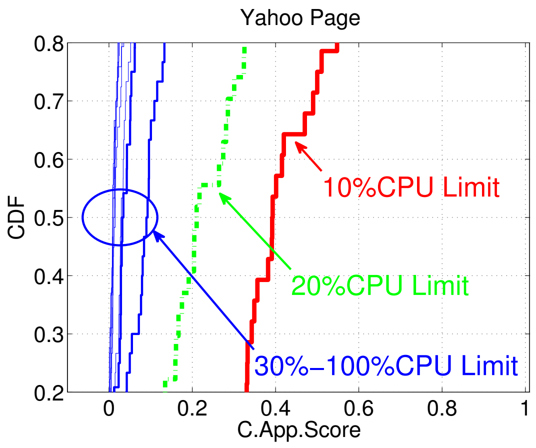


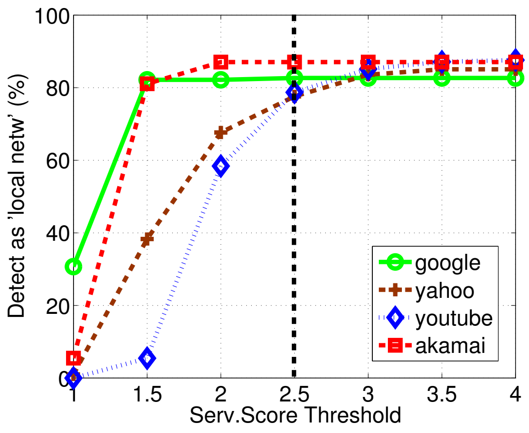
Figure A.5: Diagnosis Scheme in Flow-chart.

dans nos expériences montrent des comportements similaires, par conséquent, dans notre travail, nous choisissons le seuil de limitation du côté client de 0.2.

Nous montrons un autre exemple pour définir le seuil pour séparer le réseau des causes côté serveur (th_s in Fig. A.5). Nous avons mis en place deux ordinateurs clients qui se connectent à Internet via une liaison sur laquelle se trouve un goulot. Un PC est utilisé pour la navigation sur le Web, tandis qu'un autre télécharge des fichiers pour générer un trafic en compétition sur le goulot. Nous parcourons plusieurs pages Web populaires et explorons un éventail de valeurs pour les seuils compris entre 1.0 et 4.0. Nous calculons ensuite la fraction de sessions classée comme limitées par le réseau local. La Fig. A.6(b) montre les résultats en appliquant différents seuils entre 1 et 2: de 1 à 2, les classifications “correctes” pour toutes les pages Web augmentent fortement. En augmentant le seuil au dessus de 2.5, il est presque stable et converge autour de 80% pour toutes les pages. De ce fait, nous sommes confiants dans l’utilisation de 2.5 comme seuil approprié pour le travail de notre thèse.



(a) www.yahoo.com page



(b) Fraction of sessions classified as “local network limit”

Figure A.6: Controlled Experiments to Infer Thresholds Used for Diagnosis

Véritable Déploiement Pour valider notre outil, nous déployons également chez trois utilisateurs différents pour une durée de plusieurs mois chacun. Le Tab. A.2 résume les statistiques de base. Deux de ces utilisateurs sont en Côte d’Azur et l’autre est situé en Chine. Nous voyons aussi que tous ces utilisateurs ont accédé à un grand nombre de pages. Comme notre but est de diagnostiquer des pages Web avec des des temps de chargement élevés, nous nous concentrons sur les cas avec temps de chargement de plus de 10 secondes et nous les nommons “forte charge en temps”. Bien que ces résultats ne remplacent pas une évaluation minutieuse de l’outil dans un environnement de laboratoire contrôlé, elles permettent cependant de démontrer le potentiel de l'outil.

Table A.2: Collected Dataset from Three Users.

		Totally Browsed			Web Pages with "High Load Time"		
user	duration	$\#page$	$\#domain$	$\#object$	$\#page$	$\#domain$	$\#object$
$A(FR)$	5 month	3,451	579	501,102	808	247	142,939
$B(FR)$	3 month	1,788	263	87,898	281	114	24,406
$C(CN)$	2 month	3,766	535	317,700	466	183	63,619

Le Tab. A.3 présente les principaux résultats de la classification par notre système de diagnostic lors de l'application sur les pages "forte charge en temps". Nous constatons que pour les trois utilisateurs, il ya toujours autour de 20% des pages que sont limitées du côté client. L'utilisateur A souffre de beaucoup de problèmes de performance réseau, à la fois, par sa boucle locale et par Internet. Nous trouvons aussi pour les utilisateurs B et C que près de 40% des temps de chargement élevés sont dus à la partie serveur.

Table A.3: Limitation Causes for Web Pages with high load times

User	Main cause				
	Client	Server	Local access	Internet	others
A	**21%**	4%	**29%**	**32%**	14%
B	**28%**	**39%**	9%	10%	14%
C	**21%**	**44%**	9%	6%	20%

Conclusion de la partie II Tout au long de l'étude dans cette partie, nous nous sommes concentrés sur le diagnostic pour des utilisateurs ordinaires. Nous avons présenté notre outil qui est composé de la mesure du côté client et le diagnostic et analyse du côté serveur. Nous avons ensuite indiqué les facteurs de limitation potentiels qui peuvent limiter la vitesse du rendu des pages Web. Après cela, nous avons présenté notre modèle de diagnostic. Bien qu'il existe plusieurs paramètres à régler, nous nous sommes appuyés sur des expériences en environnement contrôlé pour souligner le caractère raisonnable des valeurs des paramètres. Après un déploiement à long terme (plus de cinq mois) chez des utilisateurs réels, nous avons recueilli suffisamment de données. En gardant les pages Web avec une mauvaise performance, nous avons constaté que (i) il y a une limitation du côté du client pour tous les utilisateurs, (ii) les mesures traditionnelles qui évaluent les comportements de réseau tels que TCP Round Trip Time peuvent parfois être insuffisantes pour expliquer un long temps de chargement. Une portion élevée des sessions de navigation sont détectés comme limitées parle côté serveur limité.

A.2.3 Nouvelle méthode d'analyse de la navigation Web

Jusqu'à présent, cette thèse à abordé différentes questions liées aux performances Web. Parmi les méthodes présentées, un problème commun est: la prise en charge de la page Web entière en tant qu'unique entité. Mais ce n'est généralement pas le cas pour yen page Web réelle. Les différents objets composant la page Web ont généralement des impacts différents sur la performance globale de cette page. Dans la partie III de la thèse, nous développons une méthode novatrice pour extraire les facteurs clés lors du téléchargement d'une page. Nous nous inspirons des méthodes du chemin critique (CPM) et nous essayons de construire un tel chemin pour le rendu de pages Web.

Méthode du chemin critique Pour construire le chemin critique, nous suivons ces trois étapes:

1. Identification de l'activité.
2. Inférence de la dépendance.
3. Extraction du chemin critique.

Nous définissons chaque téléchargement d'objet en tant qu'activité spécifique, le début et la fin du temps de téléchargement de l'objet sont respectivement comptabilisées comme heure de début et de fin du téléchargement.

Nous utilisons donc certaines caractéristiques des navigateurs Web lors du rendu d'une page et d'autres informations partagées entre utilisateurs sur la navigation de la même page Web, afin de déduire les relations de dépendance possible entre les objets. à cette étape, il peut être nécessaire d'avoir retours d'expérience de navigation pour la même page Web. Après cette étape, nous transformons le processus du rendu de page Web en un graphe acyclique dirigé.

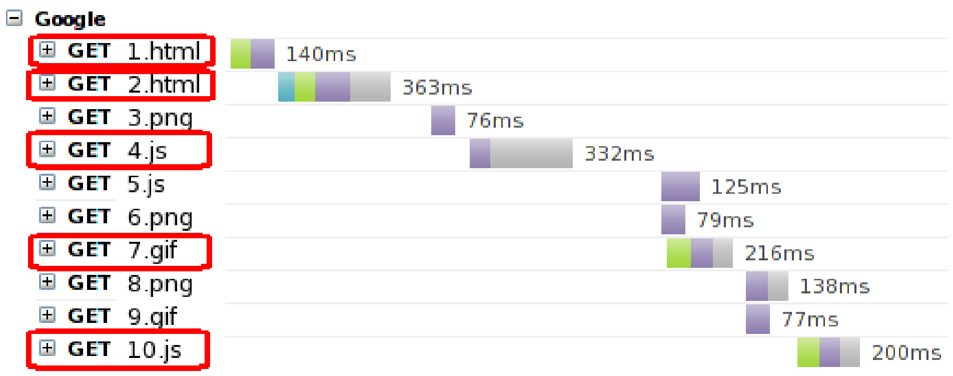
Enfin, en attribuant à chaque objet une durée de téléchargement, nous construisons un chemin critique final qui a un impact sur des performances de rendu global de la page Web.

Expérience de laboratoire Nous donnons ici un exemple typique de déploiement de notre algorithme CPM. Nous visitons en continuellement la page Web de Google avec un navigateur ayant sa cache vide. Afin d'émuler différentes conditions de navigation, nous modifions manuellement les conditions réseau des paquets sortent en introduisant des pertes et des retard avec l'outil netem. Nous configurons les paquets sortants avec quatre scénarios différents: des délais de 50ms et 100ms et un taux de perte de 10Ces résultats sont disponibles dans la Fig. A.7. Comme nous pouvons le voir sur la Fig. A.7(a) cette page se compose de 10 objets différents, comprenant HTML, Javascript et images. En analysant ces objets avec notre algorithme CPM, on obtient les résultats présentés dans la Fig. A.7(b):

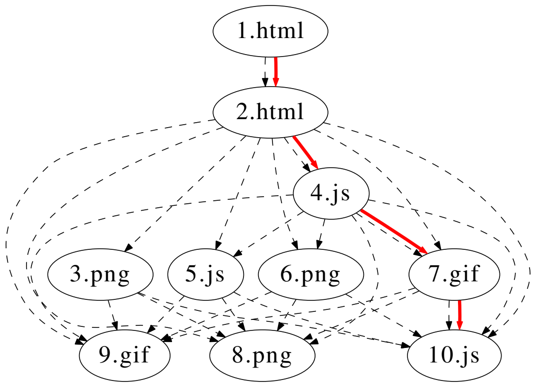
- L'objet 1.html (www.google.com) est la racine qui initialise l'ensemble de la page Web.
- Après avoir reçu les données de 1.html, la page est redirigée vers le fichier principal de 2.html qui est www.google.fr.
- Lorsque le navigateur reçoit suffisamment de données à partir du fichier principal, il commence à rendre le contenu de la page et télécharge d'autres objets incorporés soit en parallèle ou en série. Par conséquent, nous pouvons voir avec les lignes en pointillés sur la Fig. A.7(b) un lien direct pointant de l'objet 2.html à tous les autres objets.
- Selon nos observations, indépendamment des conditions réseau lors du téléchargement des objets embarqués, le fichier 4.js contenant du JavaScript peut toujours bloquer les objets téléchargés ensuite. C'est pourquoi nous voyons sur la Fig. A.7(b) qu'il existe un lien direct entre 4.js et les objets suivants.
- Les groupes d'objets 5.js, 6.png et 7.gif peuvent être téléchargés en parallèle, cependant, nous constatons qu'un délai plus long d'un objet dans ce groupe bloqu les suivants. Dans ce cas, on voit sur la Fig. A.7(b) que ces trois objets ont des liens directs vers les objets enfants suivants 8.png, 9.gif et 10.js.
- Nous pouvons également voir un lien direct entre entre les objets compris entre 3.png et 8.png, 9.gif et 10.js. Cela peut être dû au fait que, lors de nos expériences, l'heure de fin de télécharge-

ment de 3.png a une forte corrélation avec l'heure de début des trois objets et notre heuristique ne fournit qu'une estimation de la relation entre ceux-ci. En effet, comme indiqué en trait plein rouge sur la Fig. A.7(b), cette relation n'influence pas le choix du chemin critique.

- Comme nous pouvons également le voir, les objets compris dans le chemin critique nécessitent généralement le plus de temps lors du chargement de la page Google. Nous allons également montrer dans les sections suivantes que ces objets peuvent être considérés comme des objets clés de la page Google.



(a) Timeline of Object Activity



(b) Extracted Dependency Graph with Critical Path

Figure A.7: Critical Path Analysis Example for `Google` Page

Exemple d'utilisation La nature intrinsèque du chemin critique résulte à ce que "les activités de ce chemin doivent avoir un impact direct sur l'ensemble du projet". Dans cette section, nous discutons des scénarios d'utilisation de notre méthode pour les études de performances Web.

- **Estimation de la Performance**: Le premier scénario est la prédiction des changements de performances lors de l'augmentation de RTT. Pour illustrer cela, nous utilisons toujours la page Web www.google.com et effectuons des expériences de navigation similaires à celles présentées précédemment. Ces échantillons de navigation sont utilisés comme ensemble d'apprentissage afin de construire le graphe de dépendance. Ensuite, nous émulons une augmentation du RTT à l'aide netem sous trois conditions de délai de paquets sortants: 0 ms, 100 ms et 200 ms. Nous utilisons le temps de chargement de la page avec un délai de 0ms comme base et nous prédisons sa distribution dans les autres cas grâce aux étapes suivantes:

1. Pour chaque navigation dans la base, nous générons un arbre de dépendance en utilisant les ensembles d'apprentissage.
2. Mise à jour de chaque retard de l'objet dans le graphe de dépendance;
3. Trouver un nouveau chemin critique dans l'arbre mis à jour, et le retard total dans le nouveau chemin critique est estimé par le temps de chargement lors de nouveaux scénarios.

La seconde étape, concernant la mise à jour du retard du téléchargement, peut être délicate. Puisque nous utilisons une mesure grossière au niveau du navigateur, nous n'estimons l'augmentation de retard que d'une manière approximative: chaque fois que nous observons des retards sur les requêtes DNS, TCP et HTTP pour un objet, nous le mettons à jour en augmentant les délais supplémentaires; Pour les retards lors de la réception de données, nous devons mettre à jour le délai supplémentaire à chaque cycle TCP. Dans notre cas, nous estimons nombre de cycles TCP en utilisant le délai de réception divisé par médiane du délai de connexion TCP au même serveur hôte. Bien que cette estimation est grossière, nous verrons plus tard que, le temps de chargement peut être prédit avec une petite marge d'erreur pour les différents scénarios de simulation. Dans ce chapitre, nous calculons le temps de chargement comme l'intervalle de temps entre la demande de départ et la fin des derniers octets dans une page Web.

La Fig. 7.7 montre les résultats des temps de chargement de ces deux pages Web sous différents scénarios. Pour chacun des paramètres de délai supplémentaire, nous parcourons la page Web 50 fois un cache navigateur vide. Grâce à cette figure, on voit que chaque incrément de 100 ms, correspond approximativement à 1 seconde de plus au niveau du temps de chargement. Les résultats prédits sont forts similaires aux temps de chargement observés. Nous constatons que pour le cas d'un délai de 100 ms, les erreurs relatives à la médiane sont inférieures à 5%, tandis que pour 200 ms de délai, les deux erreurs dans le cas médian sont inférieures à 10%.

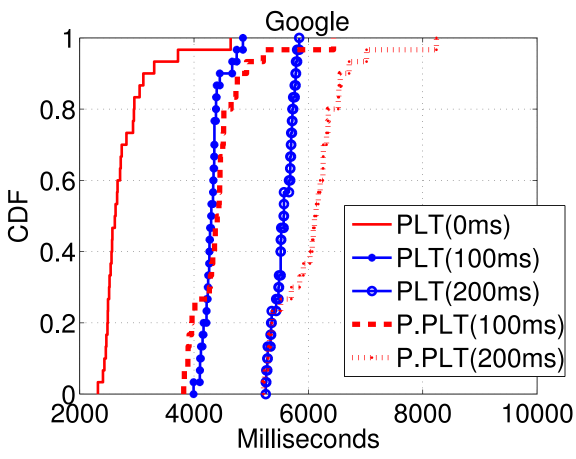
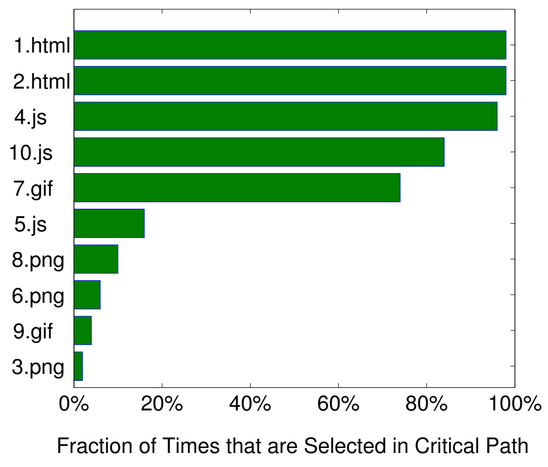


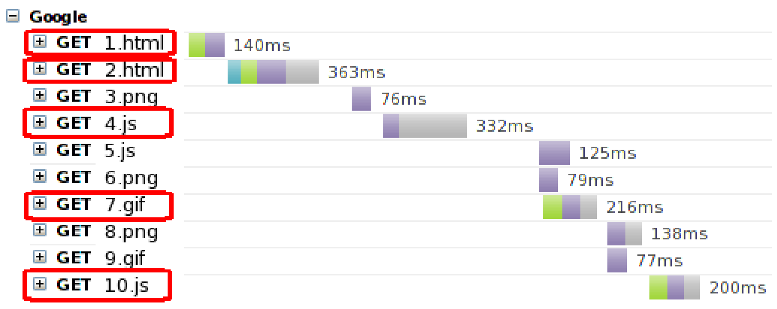
Figure A.8: How Page Load Time Changes for Google Web Page when RTT Increases (PLT=Page Load Time, P.PLT=Predicted Page Load Time)

- **Extracteur d'Objets Clés**: Un autre cas d'utilisation de notre méthode consiste à extraire les objets clés à l'intérieur d'une page. Une page Web peut contenir plusieurs objets qui peuvent avoir un impact différent en ce qui concerne le temps de rendu de toute la page Web. Nous utilisons toujours des échantillons de navigation tels qu'ils sont utilisés dans cette section. Nous extrayons un chemin critique pour chaque téléchargement de page Web. Nous comptons ensuite le nombre de fois où chaque objet fait partie du chemin critique à travers tous les téléchargements. La Fig. A.9(a) montre cette proportion.

Nous voyons que des objets comme les fichiers de script (4.js et 10.js) et HTML (1.html et 2.html) sont comptés dans le chemin critique beaucoup plus fréquemment (par exemple pour> 80% des cas), mais que les objets icône comme 3.png sont rarement critiques. Dans ce cas, nous pouvons dire que les objets qui sont souvent inclus dans le chemin critique jouent un rôle plus important pour le rendu de la page google.com.



(a) Key Objects Distribution in `Google` Page



(b) Example of `Google` Page Object Activity

Figure A.9: Key Object Extraction by CPM

Conclusion de la partie III Dans la troisième partie de la thèse, nous proposons d'utiliser la méthode du chemin critique pour analyser les performances d'une page Web. Nous transformons la vue en "cascade" d'une page web téléchargée en un arbre et ensuite identifions le chemin dans l'arbre qui affecte le temps de chargement global. Nous utilisons des caractéristiques faciles à observer mais avons besoin de multiples expériences de navigation de la même page Web.

Comme nous l'avons montré dans cette partie, notre méthode a un grand potentiel non seulement pour l'extraction des l'objets clés d'une page Web donnée, mais aussi pour la prédiction de la performance.

A.3 Conclusion

A.3.1 Contributions

Dans l'introduction de cette thèse, nous avons fait plusieurs affirmations. Nous passons maintenant en revue ces allégations et nous concluons sur les contributions de la thèse.

Les contributions de cette thèse sont:

1. *Notre plateforme initiale, qui combine des mesures au niveau de l'utilisateur et au niveau du réseau, peut réduire considérablement l'écart entre la QoE subjective de l'utilisateur (Quality of Experience) et la QoS mesuré objectivement (Quality of Service).*

   Le chapitre 2 présente notre plateforme initiale qui est basée sur un navigateur, et auquel différents autres composants peuvent être ajoutés tels que la capture de paquets, etc

   A partir de mesures au niveau du navigateur, nous pouvons connaître performances finales perçues par. à notre connaissance, nous sommes les premiers à utiliser principalement des mesures au niveau du navigateur pour étudier les expériences des utilisateurs finaux.

2. *Différents types de techniques de clustering peuvent être adoptées pour étudier les performances Web*:

   - L'utilisation du clustering classique centralisé nous permet de révéler la relation entre la satisfaction au niveau utilisateur (QoE) et le niveau de réseau inférieur (QoS) des métriques.
   - Le clustering distribué fournit une approche évolutive pour utiliser les mesures effectuées par les différents utilisateurs.

   Dans le chapitre 3, nous avons proposé d'utiliser le clustering pour révéler la relation sous-jacente entre QoE et QoS. Nous y montrons également que le clustering peut être utilisé pour comparer les performances entre plusieurs maisons.

   Partant de ce constat, nous avons étendu la méthode de classification dans le chapitre 4 pour travailler d'une manière distribuée. Nous avons créé un prototype multi-noeud pour effectuer le processus de clustering réparti entre les utilisateurs de différentes maison. Bien que chaque utilisateur possède une connaissance limitée tels que les centres de gravité et les compteurs, le regroupement distribuée était en mesure de fournir des indications intéressantes.

3. *Nous proposons des modèle d'analyse de causes profondes basés sur des mesures passives de navigation de l'utilisateur. Notre méthode est basée sur les interfaces de développement de Mozilla et stocke les données dans une base de données..*

   Dans la deuxième partie, nous avons présenté notre outil, FireLog, qui est composé de mesures du côté client et de diagnostics et d'analyse du côté serveur. Nous avons réalisé des expériences pour montrer les caractéristiques de FireLog: faible intrusion et une grande précision.

   Nous avons ensuite présenté notre modèle de diagnostic pour le dépannage. Bien qu'il existe plusieurs paramètres à régler, nous nous sommes appuyés sur des expériences contrôlées pour leurs choix.

   Via un déploiement à long terme (plus de cinq mois) chez des utilisateurs réels, nous avons recueilli suffisamment de données pour notre modèle de diagnostic. Nous avons constaté que,

(i) il ya toujours des limites du client chez les utilisateurs, (ii) les mesures traditionnelles qui évaluent les comportements de réseau tels que TCP Round Trip Time peuvent être insuffisantes pour expliquer un long temps de charge de la page, puisque la performance peut être limitée côté serveur.

4. *Le partage des expériences de navigation nous à aidé à développer une nouvelle approche dans l'analyse de la navigation de pages Web. Notre nouvelle approche, appelée méthode du chemin critique, peut être utile pour l'analyse des performances Web dans plusieurs scénarios, y compris la production de prédictions what-if et l'extraction d'objet clé.*

   Dans la partie III, nous avons présenté notre méthode, baptisée "méthode du chemin critique", pour l'étude de la performance Web. L'idée principale de notre méthode est d'utiliser les informations partagées par différentes expériences de navigation pour la même page Web.

   Nous avons introduit quelques fonctionnalités de navigation qui peuvent causer des interdépendances entre objets d'une page Web. Après cela, nous avons formalisé notre intuition et avons fourni une méthode pour extraire de ce chemin critique les objets qui affectent le rendu d'une page. Nous avons ensuite utilisé la page Web google.com, à titre d'exemple, pour expliquer en détail le chemin critique.

   Nous avons finalement proposé trois scénarios d'utilisation basés sur notre méthode du chemin critique, nommé découverte de goulot d'étranglement, la prédiction de la performance et l'extraction de l'objet clé. A partir de nos expériences, nous avons montré que notre méthode est très prometteuse et peut, par exemple, prévoir la performance de téléchargement avec 15% d'erreur pour tous les centiles.

A.3.2 Perspectives

Il ya plusieurs orientations possibles pour les futurs travaux. Nous les classons en tant qu'évolution de l'outil.

Développement d'outil Dans cette thèse, nous avons étudié les performances de navigation Web en utilisant le navigateur Firefox. Il serait intéressant d'étendre les travaux actuels en développant de nouveaux outils pour les autres navigateurs Web. Bien que Firefox soit toujours l'un des principaux navigateurs Web du marché, les statistiques [1] montrent également que la popularité de Google Chrome a augmenté à un rythme élevé, passant de 10.8% en Janvier 2010, en hausse de 46,9% en Décembre 2012. Par conséquent, le portage de notre prototype vers d'autres navigateurs peut être utile pour augmenter les mesures des utilisateurs et comparer différents navigateurs.

Il serait également intéressant de porter notre outil et une méthodologie pour les utilisateurs mobiles. Les résultats d'analyses montrent que la croissance des utilisateurs de téléphones mobiles au cours des dernières années est beaucoup plus rapide que celle des utilisateurs de bureau, et continuera probablement d'augmenter d'ici à 2015 [33]. En même temps, il est également connu que l'accès au Web via des appareils mobiles est encore lent pour la plupart des sites Web. Par conséquent, il serait également intéressant d'adopter notre prototype dans le domaine mobile.

Développement du modèle Une autre direction pour le travail futur peut concerner des modèles de diagnostic plus intelligents. Bien que nous ayons proposé un modèle de diagnostic dans cette thèse (chapitre 6), il y a encore des limitations. Par exemple, un modèle de collaboration entièrement automatique peut être développée en tant qu'amélioration de notre prototype actuel. Garcia et al. [55] proposent un système de collaboration pour le dépannage de réseau basée sur agents multiples et le réseau bayésien; des architectures similaires sont également proposées par le système DYSWIS [70] et KOWLAN [54].

Comme nous l'avons vu aussi dans la thèse, des corrélations existent entre les grands retards et l'insatisfaction des utilisateurs. Cependant, la corrélation n'implique pas nécessairement causalité. Il y a, par exemple, de nombreuses raisons pour expérimenter des retards importants tels que un grand buffer à la liaison d'accès ou un proxy surchargé entre le client et le serveur. Il serait intéressant d'utiliser le cadre développé par J. Perl [66] pour étudier les relations de cause à effet.