JOHANNES KEPLER UNIVERSITY LINZ | JKU

TNF

Faculty of Engineering and Natural Sciences

Web Performance Monitoring and Tuning

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements
for the academic degree

Bachelor of Science

in the Bachelors Program

Computer Science

Submitted By:
Jürgen Ratzenböck

At the:
Institut für Telekooperation

Advisor:
Gabriele Anderst-Kotsis

Linz, 20. November, 2015

Affidavit

I hereby declare that the following dissertation ”Web Performance Monitoring and Tuning” has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Linz, on February 9, 2016

Jürgen Ratzenböck

Acknowledgment

Hereby I would like to thank my supervisor Univ.-Prof. Mag. Dr. Gabriele Anderst-Kotsis for her support throughout the whole work on this thesis. She always provided me good and useful feedback and used her professional skills and excitement about this topic to give me hints and advices when I needed it. Without her support this thesis would not have been possible.

Abstract

There have been dramatic changes over the last years regarding the web, it's content, purpose and complexity. Websites are not just simple HTML files anymore which reside on a server at a single location. Many businesses operate highly available web applications which often provide a lot of functionality delivered anytime to users around the world. Customers expect those websites to be fast, otherwise they just leave. Several studies showed that good web performance is a "must-have" feature in modern and beloved web applications and increases chances of attracting potential customers evidently.

The requirement of high performance adds another activity to the multi-disciplinary field of web development. This work explains the important factors Latency and Bandwidth dictating the time messages spend on the way between client and destination server. Some examples outline the high influence of the distance requests and responses have to travel, how Latency and Bandwidth correlate and where bottlenecks possibly occur. Following that the browser has to process incoming responses and build up the web pages as the user finally perceives it which of course introduces further delays. The correct and intelligent usage of HTML, CSS, Javascript and multimedia content affects user experience positively.

For the development of the new Website of the pet tracking company Tractive the current system was monitored with several tools to find out potential bottlenecks. Different load tests gave some indication of how the servers and the backend performs under stress while browser processing investigations were leveraged to detect blocking and performance critical resources. Finally some suggestions to tackle existing problems are presented which give an idea of how some things can be improved or re-implemented to support high performance.

Contents

Abbreviations

CSS Cascading Style Sheet

HTML Hyper Text Markup Language

SSL Secure Socket Layer

HTTP Hyper Text Transport Protocol

HTTPS Hyper Text Transport Protocol over SSL

GPS Global Positioning System

CPU Central Processing Unit

DOM Document Object Model

CSSOM Cascading Style Sheet Object Model

ERB Embedded Ruby

JSON JavaScript Object Notation

REST Representational State Transfer

List of Figures

List of Tables

Chapter 1

Introduction

In 1990 the first website was published on the Internet. Although designed for world wide availability, the actual number of users who had access to the Internet was rather limited at these times. Over the years the Web has evolved dramatically with respect to content volume, media diversity and user population. Nowadays the internet is nearly inevitable for successful world wide businesses. Companies present themselves with websites, sell their products and provide services for their customers online.

A good web experience is therefore crucial and the speed of a website often plays a key role whether an interested visitor becomes a customer or not [7]. Therefore it's a factor a successful company should not neglect when developing and maintaining a website. This thesis documents the detailed investigation of the performance of a business website and outlines possibilities to improve the perceived web experience based on the Upper Austrian pet tracking company Tractive GmbH.

1.1 Motivation

Throughout all the innovative developments in telecommunication the web browser might have become the most widespread deployment platform to developers. Industry growth projections [3] predict an amount of 20 billion connected devices by 2020 and each device comes with a browser installed. In contrast to all different applications which are somehow tailored to fulfil some defined tasks for a given targeted audience, the browser does not have any limitations disregarded of the type of the device, used platform, manufacturer etc. Due to this importance the browser from the early days looks nearly nothing like the one we use nowadays on our devices. Concurrently with these technical innovations the variety and complexity of tasks which can be done via the Web has increased as well and the user expectations on web applications are very high. This is why web performance definitely has to be considered a feature next to design for instance. Empirical studies showed that users tend to drop away from slow websites whereas faster websites lead to

- better user engagement
- better user retention
- higher conversion rates (indicates how many of the website visitors convert the paying customers)

Single page applications, responsive designs, cool CSS3 animations or a bunch of new Web UI components featured by HTML5 are only a few things which come along with modern web applications as they appear in our todays web browsers. All these innovations add complexity and therefore more data to be handled throughout the network. This introduces a further challenge for web engineers as all this design tweaks require more work to deliver high performance web applications to their users. Understanding how the web with it's different underlying protocols works is essential. Good developers have to have some deeper knowledge of how the bits within a HTTP request are transported from one point to another as well as how the browser on the client side renders HTML, loads media and processes JavaScript [3].

1.2 What is Tractive?

Tractive GmbH produces hardware devices for tracking pets and develops software applications which allow users to communicate with their pets wearing these devices. The company was founded in Pasching, Upper Austria in 2012 and offers today several different devices whereby the origin goal was to find and locate pets fitted with a GPS tracker using a Smartphone App. In addition to the hardware devices and mobile apps Tractive develops and maintains a Website providing all the necessary product information for potential customers as well as a web shop which allows customers to purchase products and a web application where registered users can manage their trackers and pets.

The company is interested in analysing and improving the performance of their web site, including all de-facto static content as well as the mentioned Pet Manager. Relevant components are to be re-designed as well as integrated and connected to the current backend containing the data and business services used by client applications. Next to a good "look and feel" it is important for the company to evaluate how well the current website performs and try to find bottlenecks which should be considered and fixed when integrating the new Website.

1.3 Objective and Outline of the Thesis

This thesis will contribute towards a deeper understanding of performance influencing factors of modern web sites. It will therefore start with an overview of web performance metrics typically used in performance evaluation studies (see section 2. To relate those metrics to the actual performance behaviour of a real world web site, the tractive case study will be used. In section 3 the architecture of the system under test will be briefly described. The next section 4 will discuss the experimental design of the measurements as well as the results. Based on this, suggested improvements 5 and an outlook on future work **??** will conclude the thesis.

Chapter 2

Web Performance Metrics

Before any measurement or even improvement regarding a website's performance can be made, it's important to understand what actually are the reasons for the delay until a user can see the rendered webpage in her browser. Since a website is not like an

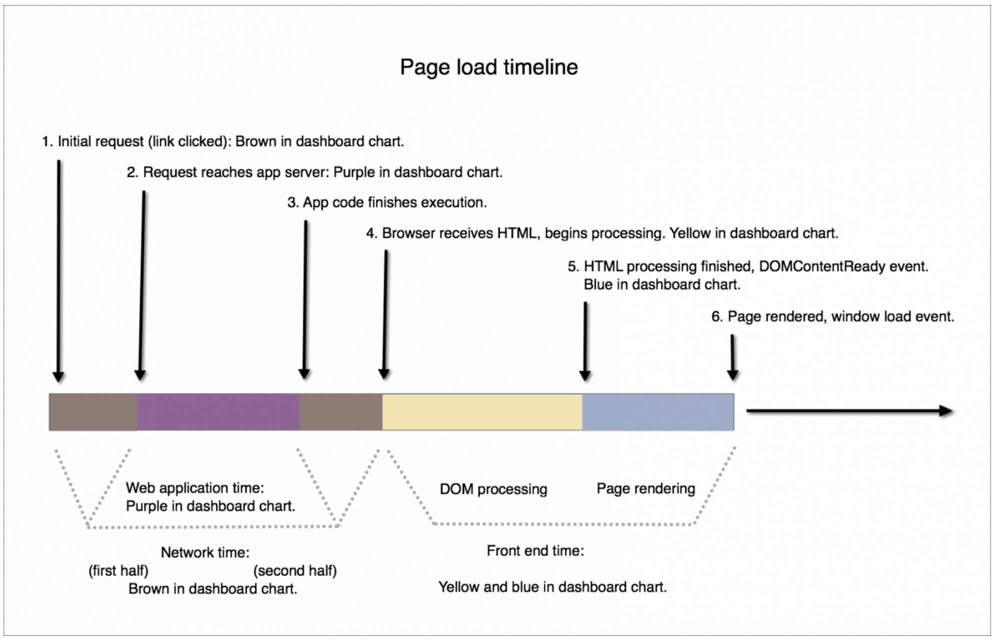
Page load timeline
1. Initial request (link clicked): Brown in dashboard chart.
2. Request reaches app server: Purple in dashboard chart.
3. App code finishes execution.
4. Browser receives HTML, begins processing. Yellow in dashboard chart.
5. HTML processing finished, DOMContentReady event. Blue in dashboard chart.
6. Page rendered, window load event.
Web application time: Purple in dashboard chart.
DOM processing
Page rendering
Network time: (first half) (second half) Brown in dashboard chart.
Front end time: Yellow and blue in dashboard chart.

Figure 2.1: Timeline of a page load [8]

executable started at the computer of a client, there are several indicators affecting the overall performance of a website at which I want to look at in more detail now.

2.1 Latency

Latency basically is the time it takes a message to travel from it's point of origin to the destination. Although this sounds very simple, it's clear that not every source in the network is directly connected to a destination point. The global network infrastructure is much more complex and therefore several components contribute to the overall latency. In essence latency can be grouped into:

- Propagation delay
- Transmission delay
- Processing delay
- Queuing delay

Propagation delay is the time required for a network packet to travel from sender to receiver. It simply can be computed as a function of distance over speed and is heavily based on the speed of light, the maximum speed at which information can travel. Fortunately the speed of light is extremely fast and even including a small loss of speed due to refraction of the medium through which the message travels it only takes a few hundred milliseconds to cross the distance between two continents. There have been a lot of improvements regarding the medium connecting network components and with fiber-optic cables a speed of approximately 200.000m/s can be achieved nowadays [3].

Route	Distance	Time (vaacum)	Time (fiber)
New York to San Francisco	4.148km	14ms	21ms
New York to London	5585km	19ms	28ms
New York to Sydney	15993km	53ms	80ms
Equatorial circumference	40075km	133.7ms	200ms

Table 2.1: Examples for propagation delays [3]

Transmission delay is the time it takes to put the packet to be sent onto the data link [3]. The bytes to be sent over the network are not automatically on the link and therefore there must be this delay. Due to the small size of request and response packets as they are used for websites, this amount of time usually stays within a few milliseconds. It can also be described as a function of packet length divided by data rate of the link.

The latencies presented the table above assume a direct connection between two locations. In practice this does not hold since a packet gets handled by several intermediate routers until it reaches the destination. Each router must accept the incoming packet first, look at the header information of it and decide where to send the packet next. The time required for these operations is called **Processing delay**. This delay can vary from packet to packet and strongly depends on the number of hops a packet does on it's way [5]. Nearly all of the operations are done in the hardware and cost less time [3].

Due to everyday's enormous amount of traffic the network can be unpredictable under some circumstances. If the traffic load on routers is high some packets have to be put into an incoming buffer and will be queued before they can be processed. This is known as **Queuing delay** and can add some additional time to overall latency. The concerned routers on the way usually vary in latency and bandwidth as well as the traffic along the path which makes it difficult to predict occurrence of queuing.

The sum of these delays results in the total network latency and under normal circumstances does not exceed about 300ms, even when crossing oceans and continents [3]. Although these delays don't seem to introduce any kind of problem, especially when we consider such extraordinarily large distances like in the example, the parameter latency is critical and can introduce a performance bottleneck. This will get clearer when looking at the involved protocols by a request.

2.2 Throughput

Throughput is the number of items processed per unit time, such as operations per minute or instructions per second. When talking about network connections the term Bandwidth is usually used instead, which is equal to the throughput in bits per second. Regarding bandwidth there hast be differentiated between the core network and the edge points.

Bandwidth in core networks is the bit rate per seconds for links across long distances, like subsea connections between continents. Different cable mediums have been researched over years whereby with nowadays capabilities of fiber-optic connections, amazing rates of 70Tb/s for a single fiber-optic link could be reached [3].

On the edges of the network things look completely different. **Bandwidth at edge points** is the data rate we know from our ISP and depends heavily at the technology used. No matter whether it is dial-up, DSL or fiber-optic to the home these numbers are very small compared to the core network bandwidth [3].

Bandwidth is not directly connected to Latency, which means that a low latency not necessarily implicates high bandwidth of the data link. However if we take the bandwidth at the network edge, an increase of maximum throughput often leads to lower latency since more packets can be processed in the same time and collisions and congestion will more likely be avoided. If the problem occurs at some intermediate hops on the path, higher bandwidth does not have any positive effect on latency [5].

2.3 Utilization and Efficiency

These two performance parameters give a more general indicator of the system's components performance. **Utilization** is the fraction of capacity of a component used [5]. This gives a good hint how busy a server component currently is and how much load it can handle by operating still normal. There are a lot of common utilization performance metrics which can be measured. The most popular ones are

- CPU
- Disk
- Memory

Although these metrics are not directly connected to the performance a user experiences when she waits for a web page to be visible in the browser, it helps to check whether infrastructure components operate as expected. The reason for bad performance is not always based on poor latency or throughput, sometimes whole components does not scale as desired and this can have negative impact on what happens on the server side.

Efficiency is defined as throughput divided by utilization. Efficiency cannot be measured but can reveal how much components can withstand. This metric can further have some influence in economic decisions when deciding about buying resources. In this case it is more likely known as cost-effectiveness. [5]

2.4 Processing Time on Server

This is the time between the arrival of the request on the server and the point where the server starts to send the response. For the performance of a web application it is an important metric as it gives information how fast the server fetches and builds the web resource requested. Since modern websites don't contain only pure HTML code usually much more work has to be done to generate the outgoing response file. This includes algorithms to be executed, working with databases in the backend, pre-processing CSS and Javascript extensible languages, loading of images, generating HTML and so forth. Interactive web applications which work with bigger datasets can lack of performance pretty fast if you don't pay attention on using appropriate software technologies or doing cost-intensive work to fulfil interactive tasks of a user.

2.5 Browser Processing

When the first bytes have arrived at the client it's time for the browser to start it's work. Obviously it is important for good user experience that the time span between a request and the arrival of the first byte should be as minimal as possible because during this time nothing can happen inside the browser. Similar to the networking and processing time there is potential to save time and improve user experience when making the website visible.

2.5.1 DOM Processing

The first step the browser has to do is to identify the fetched resource from the server, where in it's simplest form it is HTML content. The received characters are disassembled into tokens and the DOM (Document Object Model) is constructed. The DOM is required to render any element on the screen and can be blocked by some other resources and therefore delayed. The same happens with CSS applied on HTML elements. For this purpose a so called CSSOM (CSS Object Model) gets constructed and afterwards combined with the DOM to a render tree ready for painting. Due to the reason that a website is effectively unusable without any style, CSS is basically treated as a render blocking resource and interrupts processing of other content. Another blocking resource is Javascript no matter if placed inline or within an external file. DOM construction will be interrupted if Javascript has to be executed. Processing the final DOM for first page view early shortens the time until elements appear on the screen and therefore this time point is a crucial one regarding any performance optimizations during web development. [2]

2.5.2 Page Rendering

After the DOM and CSSOM have been constructed and composed to the render tree the elements can be painted onto the screen. Although much of the work is done automatically by the browser, there are some performance tweaks possible to significantly improve user experience. As metric which can be directly measured it is usually referred to the final onLoad-Event indicating the completeness of a page load when every element of the webpage is visible on the screen. However, most often it is more important to improve the visual appearing of single elements like text, images, multi-media content etc. There is a huge difference for the user whether one essential page element is visible after 0.3 seconds and the following after 0.6 seconds or both elements appear after 0.6 seconds. Page rendering is an important metric but has to be observed differently since here perceived experience is critical.

Chapter 3

Architecture of the system under test

Due to steady evolvement and increasing requirements the architecture of the system under test has become quite challenging. Physical resources and several software services have been added over the last months. Used software technologies and integration of Tractive's main website and dynamic web application into the whole environment as well as other server components and software tools influence web performance. The following paragraphs give an overview about the infrastructure concerning the web applications and other relevant services.

3.1 Infrastructure

Due to high availability and scalability requirements the main backend services and databases as well as the web application run on dedicated root server instances provided by a web hosting provider located in germany. The hosts are classified into three datahosts containing our databases (MongoDB, Redis) and three multi-purpose application hosts. Regarding basic performance measurements of the web application the multipurpose application hosts are relevant. They are optimized for user and hardware traffic and have the following technical specifications:

- tractive02 and tractive03: Intel i7-2600 Quad-Core processor, 32GB DDR3-RAM, 2x 3TB HDD (Software RAID1)
- tractive04: Intel Core i7-920 Quad- Core processor, 48GB DDR3-RAM, 2x 2TB HDD (Software RAID1)

For the backend services as well as the current web application we use the full-stack web framework Ruby on Rails which provides many nice features to build a RESTful web application. Therefore the multi-pupose application host is mainly a Rails Application Server. Besides we have other software components which support us to provide a highly available and scalable web application illustrated in the picture below.

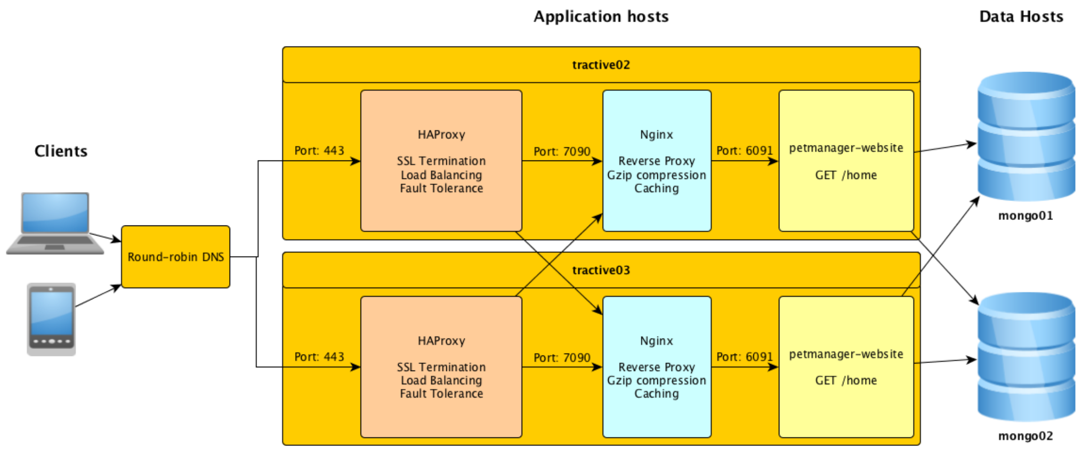

Figure 3.1: Illustration of infrastructure and platform (simplified with only two application and data hosts)

Since DNS can hold multiple IP address records for the same domain name it is possible to redirect incoming requests using a Round-robin DNS technique. This algorithm selects the IP addresses of Tractive's three application in an alternating way. A core component of the infrastructure is the HAProxy (High-Availability Proxy) which is the first receiver after an IP-address selection. Incoming requests independent of the type of the service they concern get distributed equally throughout the available servers. In addition to load balancing the HAProxy does health checks and prevents forwarded requests to unavailable services on an instance. Instead such requests are forwarded to a healthy instance. Furthermore HTTPS requests are terminated appropriately and custom ports (protected from outside) are used from this point on. Behind there is a Nginx web server as a powerful component mainly used for caching content, serving assets, providing compression and forwarding requests to the correct service acting as reverse proxy. Especially for static content it allows to return from here directly back when a cache hit was successful. Otherwise the requests gets proxied by and the according service, which for the web application is the petmanager-website service does it's work. In case user-related data is necessary backend algorithms might be triggered and probably the data hosts are required. The petmanager-website collects the necessary information and generates the requested web page dynamically. So far a quick overview about the overall infrastructure and it's components before I give some insights into the relevant services.

3.2 Relevant Services

Several different services within the backend provide the necessary functionality to the end-user applications. To understand what actually happens on the server, I want to give a short overview on the relevant services which are involved in the measurements illustrated in the following chapter. The listed services have a Ruby on Rails Application as it's fundament and use different third-party modules and dependencies to improve productivity and development. These dependencies are called gemsets in a Rails application and are installed and managed via the ruby gem dependency manager Bundler.

3.2.1 Petmanager-Website

The Petmanager-Website contains on the one hand the whole static website presenting the company itself, it's hardware products, apps and provides a support center for customers and on the other hand a dynamic web application (PetManager WebApp) where registered users can login and manage their subscribed trackers, assigned pets, subscriptions and other settings. Furthermore they have the opportunity to live-track their pets on an integrated online map. All the views are implemented via ERB (Embedded Ruby) Templating which allows generating a HTML page by filling in the dynamic information which comes as response from the business logic. Before these web pages get generated an incoming requests gets mapped to a specific route and an action within the concerned controllers is triggered. It collects the required data from the business logic in the backend and invokes page generation. Other resources are assets like Javascript, CSS and images which are primarily served and intelligently cached by the Nginx web server.

3.2.2 Platform-Graph

This is the central service and interface between the end-user applications, for instance Smarthpone apps and website, and the data tier. The service is completely written in Ruby and communicates with the Petmanager-Website via the controller classes in case of the dynamic web app. Therefore it retrieves the queried results from the data backend and exports it in a JSON format. The most important things are user related data, trackers, pets, notifications and GPS positions.

3.2.3 Platform-Common

Platform-Common represents the model in Rails MVC architecture and is responsible for retrieving data from the MongoDB and passing the aggregated results to platform-graph service. All persistent database elements stored in MongoDB collections are mapped to ActiveRecord Ruby classes which allow to query the database using a MongoDB driver. The whole logic and algorithms using this data is implemented within platform-common. Although this service has nothing to do with any HTTP request or response it affects the Application performance of the PetManager Web-App in case of slow database queries, missing indexes or cost-intensive resp. bad implementation of algorithms.s

Chapter 4

Measurements

This chapter represents the main part of the practical work. Based on the identified metrics several measurement methods and tools are explained and the results shown and analyzed. Starting with the measurements of the time passed between the user entered the URL into her browser's address line and hit the "Enter" button and the receiving of the critical first byte on the client side we are also looking deeper into the browser processing.

4.1 Leverage of fundamental Internet Protocols

As the discussed metrics can have a huge impact on the overall web performance, we take a short look on the fundamental internet protocols involved when requesting a web page.

HTTP is the main protocol responsible for delivery of a request to the server and a response back again. The method and relevant information is packed within a so called HTTP header and body. Request methods supported by every browser are GET and POST whereby GET allows to send custom parameters (data concerning the purpose of the application) within the header and POST within it's body. Furthermore the browser can add different properties to tell which version it supports, whether compression and caching can be used or TCP connections should be kept open. Adapting these properties in a web server's configuration can help to optimize performance remarkably [1].

TCP is the basis for nearly all of the HTTP traffic delivered on the Internet. Changes by web engineers in HTTP configuration can influence the performance of TCP and the underlying network. TCP is very sensitive to latency as connections start with a three-way handshake between client and server to agree on connection specific variables. Since TCP is a reliable protocol it contains special algorithms to control and avoid congestions as well as handle bandwidth differences and network condition varieties appropriately. In general TCP is conservative about packet sizes to avoid overwhelming the other side and the underlying network. As a consequence transferring a bigger HTML file can easily lead to several roundtrips between the two stations [3].

SSL represents the security layer for HTTP requests. For several years now more and more HTTP connections are encrypted with SSL to provide encryption, authenticity and integrity. The same is the case for Tractive's web application. Despite these advantages SSL contributes to network latency a quite big part since establishing of a SSL Tunnel requires a so called SSL handshake which diminishes network latency. First Client and Server have to advertise their CipherSuites and other encryption details. In a second step they use public key cryptography to negotiate a shared secret key without any prior knowledge of each other. Then the client (and sometimes also the server) checks the identity of the other side based on verified certificates and integrity of the messages are checked via a checksum too. In total this handshake requires two full roundtrips and therefore another bunch of hundreds of milliseconds. Unoptimized SSL settings for HTTP connection can lead to performance problems when a lot connections have to be established.

4.2 Comparing HTTP and HTTPS requests

The goal of the first measurements is to split a basic HTTP request into it's single parts and to get out how much time each of them costs. CURL is nice and simple REST command line tool which comes with some useful options to analyze the performance of a specific request. A bash script including a custom output template provides some insights into the concerning time periods.

Time period	Time
DNS Lookup	5ms
TCP Connect	36ms
Application Connect	0ms
Pre Transfer Time	**36ms**
Redirect	0ms
Time to first Byte	60ms
Total Time	60ms

Table 4.1: Requesting tractive.com/en over HTTP

As we can see from the results the time spent on the network is just a few milliseconds here because the network latency is not that much due to the pretty short distance

Time period	Time
DNS Lookup	4ms
TCP Connect	28ms
Application Connect	**179ms**
Pre Transfer Time	**179ms**
Redirect	0ms
Time to first Byte	264ms
Total Time	321ms

Table 4.2: Requesting tractive.com/en over HTTPS

between our office in Pasching and the Servers in Germany. Despite it takes 29ms in the first test to establish the underlying TCP connection. The main page which was requested here does not require much work on the server since only the HTML page using the correct language file for the text (English in this case) has to be generated by the Rails application server. This happens in moderate time in both cases. Fortunately the bandwidth in the office network is quite high and the Wi-Fi connection stable so the content is downloaded really fast. What makes a big difference is the use of an SSL Tunnel for the HTTP request. Although the propagation latency is just a few milliseconds due to the short distance the two extra roundtrips as mentioned earlier lead to a noticeable drawback. Luckily the evolvement of SSL improved the handshake procedure a lot so that it's not necessary for instance to do the same algorithm for every established TCP connection.

4.3 Measuring with Apache Benchmark

CURL is a convenient tool to get some quick information about REST calls and enables us to look at some performance indicators. However, for further performance tests it is not well suited and therefore the Apache Benchmark (shortly called AB) is my choice. It's a command line tool as well which gives a rich set of information out-of-the-box and can be effectively used in custom performance scripts. Since scalability for well-visited web applications is essential for any successful web business the following tests try to figure out how the involved components and the network time gets affected under increasing load by comprehensive measurements. This method is also called load testing.

Example for a simple ab test triggering 100 requests whereby there are 10 concurrent ones using HTTP keep-alive to perform multiple requests within one HTTP session.

```
ab -kn 100 -c 10 https://tractive.com/en/
```

The following parts look at some of the measurement results in more detail. Therefore I will show and analyze primarily the time periods named connection, processing, waiting and total time to first byte.

Connect: Network latency plus overhead to establish connection to remote host which includes establishing DNS lookup, TCP connection and SSL handshake. **Processing:** This is the server response time. The time it took the server to process the request and send a reply. **Waiting:** Time between sending the request from client to server plus time after processing until receiving of first byte. **Total:** Sum of Connect + Processing which equals the Time to first byte.

4.3.1 Sequential requests of different pages

First of all I compare the sequential requests of the page which shows the Tractive mobile apps and the page which shows the Tractive tracker devices.

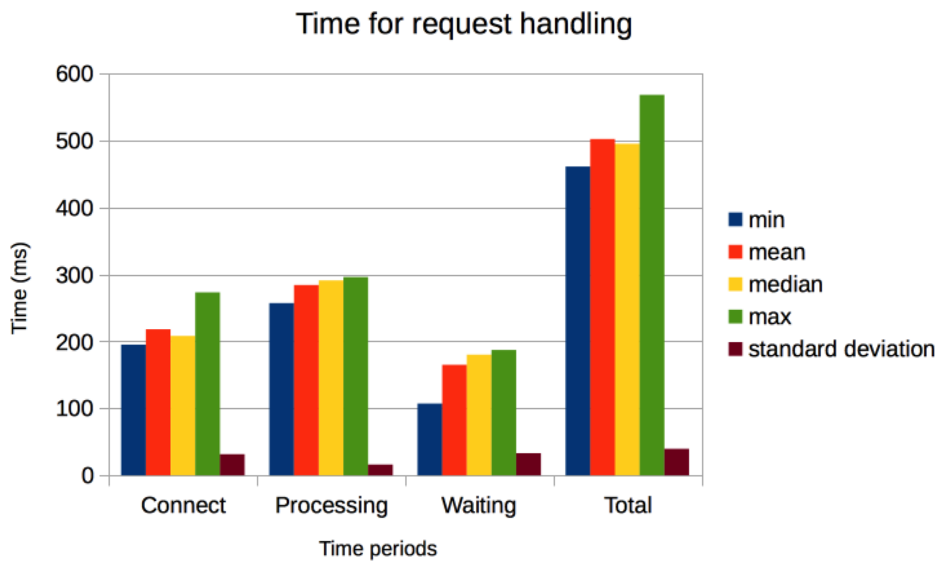

Figure 4.1: Five sequential requests to https://tractive.com/en/apps

As we can see from the results, the processing and waiting time mainly depends on the type and content of the page. The apps page requires due to its nature more processing work on the server than the trackers page and therefore there is this difference. The connection time in contrast stays pretty consistent at a median of approximately 215ms. This does not change with content or processing time on server, it depends on latency. This fact furthermore emphasizes how important network latency is to provide a fast Time to first byte.

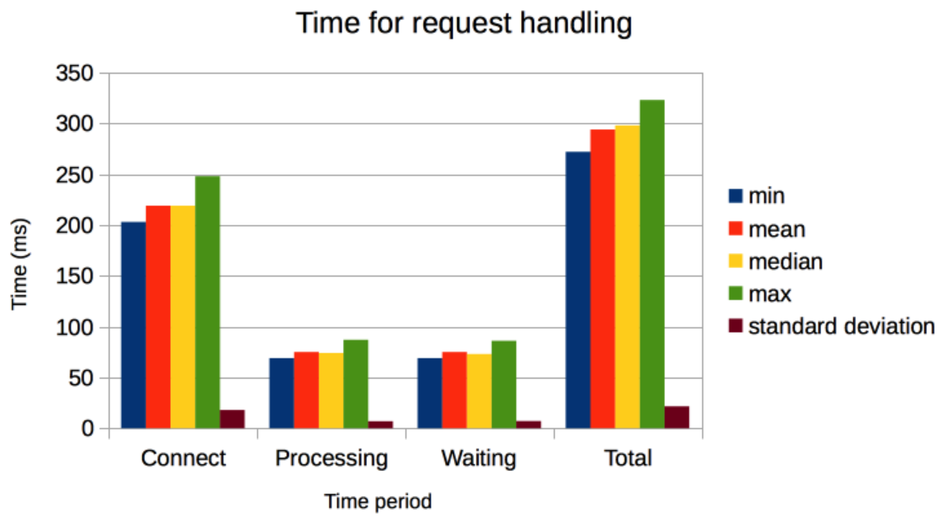

Figure 4.2: Five sequential requests to https://tractive.com/en/trackers

4.3.2 Load testing with concurrent requests

With concurrent requests you can stress the web servers as well as the underlying network including intermediate routes on the way. Although I did some measurements with a rather unrealistic number of concurrent and sequential requests for the same page it's quite interesting to see how it affects the time indicators. The pictures below illustrate performance under heavy concurrent load and shows the development in case of increasing concurrency.

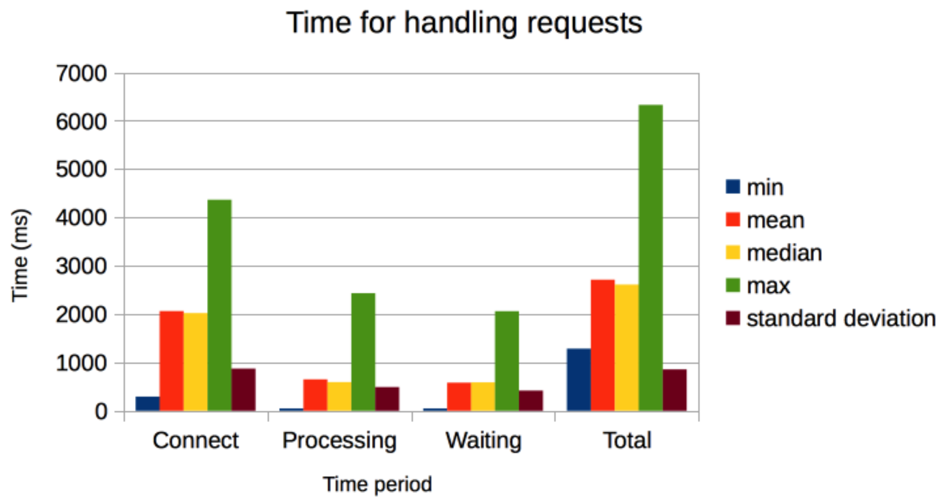

Figure 4.3: Requesting https://tractive.com/en with 1000 requests - 100 concurrently

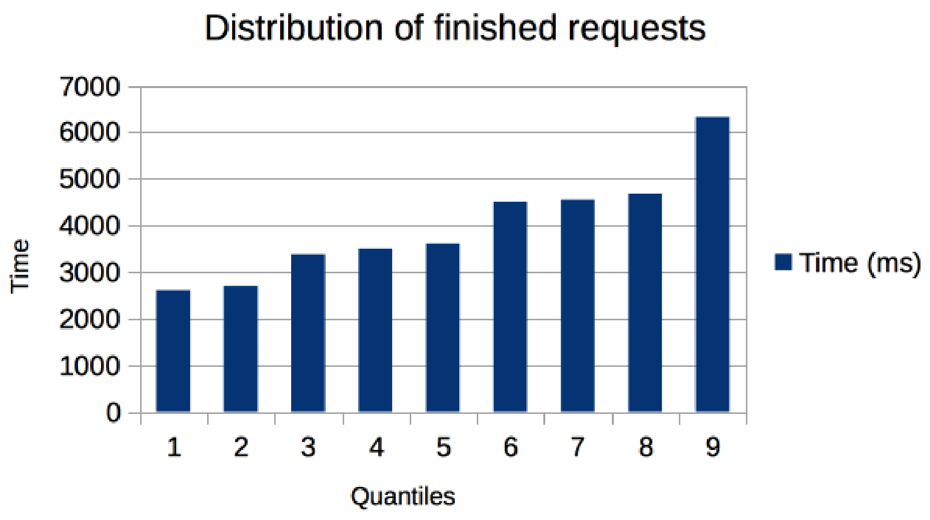

Figure 4.4: Distribution of processed requests

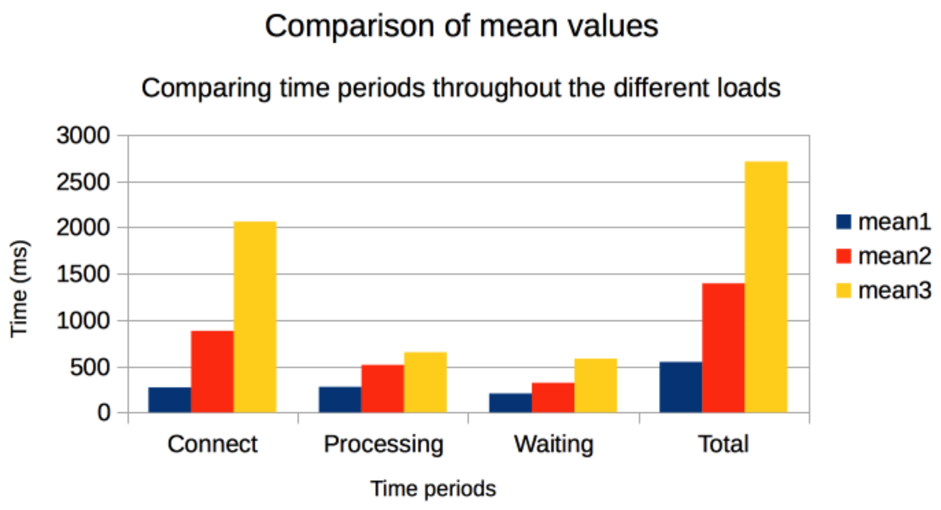

Figure 4.5: Comparing mean values for increasing concurrency resp. amount of requests

The results show that the standard deviation becomes really high especially for the connection setup time which means that the network is unpredictable. Most likely packets pile up at intermediate hops and get queued or TCP connection set-up slows down due to the enormous amount of requests in such a short time window. Whereas the standard deviation for the processing time remains lower. The redundancy of three multi-purpose application hosts with the HAProxy as load balancer takes some load off each instance. In the distribution graph you can see that it takes quite a long time to finish the last percent of requests because there are some extreme outliers. The metric

$$\frac{median}{minimum}$$

gives a good indicator whether data points are clustered around the median, the server is operating normally and can handle requests with similar balance. The optimum would be a result around 1 which in this case is absolutely not the case. The last graph tops the load testing with concurrent requests off by showing the negative development by increasing load. Tractive uses New Relic as it's main application monitoring system. All services can be monitored effectively within this system including alarms when thresholds are exceeded. It also gives some insights into performance of the website and figure 4.6 shows the impact on network time during one of my load tests.

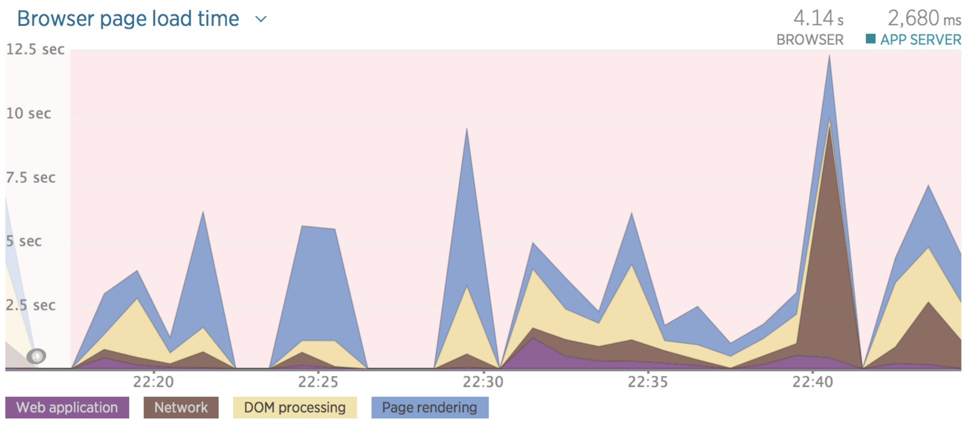

Figure 4.6: Web Performance Monitoring with New Relic

4.3.3 Consistency of performance results

This test should analyze whether the total time (sum of connection setup and processing time) stays consistent among requests or if there is any unexpected outlier. A customizable bash script allows to fire a fixed amount of sequential requests in regular time periods. Following there is a short code snippet.

```
for i in ‘seq 1 $1‘
do
  ab -kn $2 https://tractive.com/en
  sleep $3
done
```

To improve readability of the graph the executed performance test considers the total time of 15 requests of the same page each delayed by 5 seconds.

Except the first request the total times are pretty consistent around 300ms which is good. The processing and waiting times are pretty equal throughout all requests. This indicates that the server can handle the tested sequential requests without any

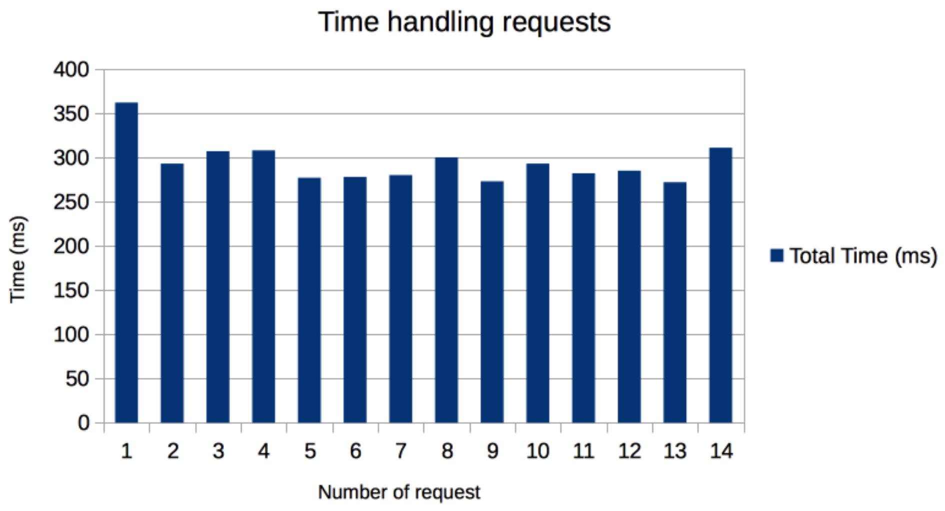

Figure 4.7: Conistency throughout the requests

problems over the provided time frame. Only the connection time of the very first request is approximately 50-70ms longer than the connection time of the successive requests. This behaviour is not completely clear but I suppose this is due to a DNS lookup at the first request which does not have to be done for the following requests.

4.3.4 Performance regarding daytime

Since page load time and especially the processing of a requests also depends on the current load of the server as we have seen from previous load tests. Customer activity can also vary throughout the daytime and regarding the Tractive Web application it can be assumed that most users are online during the day since the dominant user basis is within Europe. The following measurements should find out whether this fact actually has an impact on the network and server performance. The results of the performance test are compared and displayed in the graph below.

Although the results are pretty similar the processing and waiting time is less during the night supporting the initial assumption. However, for a secure prove of this fact other metrics, such as the current number of users at the concerned time, have to be considered and several measurements under similar circumstances should be taken.

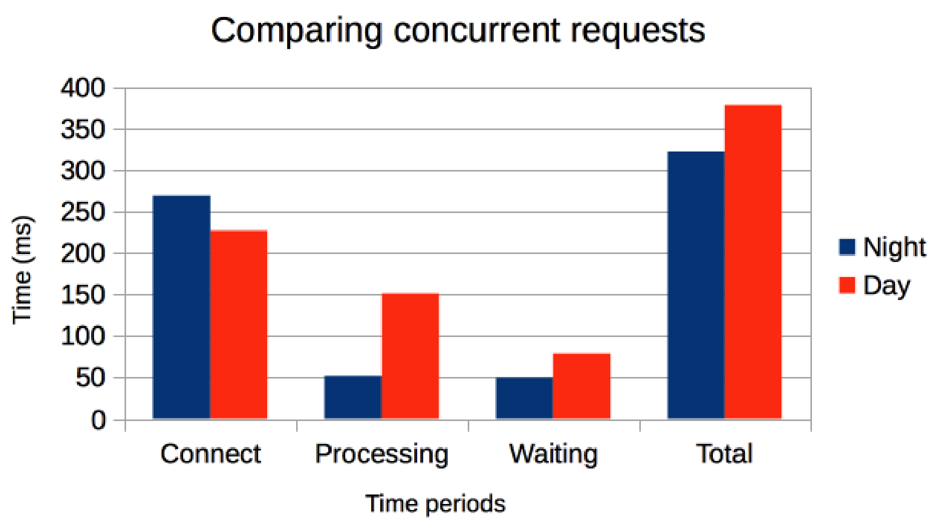

Figure 4.8: Comparing 100 requests - 10 concurrently during day and night (10:25am, 11:15pm)

4.4 Measure Browser Processing

We heard a lot about network latency, time spent establishing connections and handshakes. This section shows the second big part of my measurements which is the performance of downloading resources like HTML, Javascript or Images and displaying the result in the browser. As mentioned in section 2.5 there is usually a lot of work for the browser to render the page successfully and web developers can optimize some things here to improve the user experience. webpagetest.org is a rich tool to analyze the web performance. It gives deep insights in the building process and reveals possible problems or bottlenecks. Since Tractive is a globally operating company it is very interesting to see how the performance behaves when requesting it from the USA or even Australia. I considered this in my tests as well to show how it affects the loading times and emphasizes possible problems. Figure 4.9 shows the results when requesting the main webpage from Sydney, Australia.

First of all the results provide an overview on important metrics starting with **Load Time** which indicates the duration until the page is completely rendered and no further work has to be done. The **First Byte** is the well discussed time when the first byte has arrived on the client. Until this point of time the browser can't do anything and just **Start Render** indicates when the browser starts rendering the first element on the screen. It is crucial because until this point of time the user starres at a blank page and just sees the circling spinner. We can see from the second table that prior to Start Render the domContentLoaded event is fired and therefore empasizes how important it is to get the DOM ready as soon as possible. The **Visually complete** time is

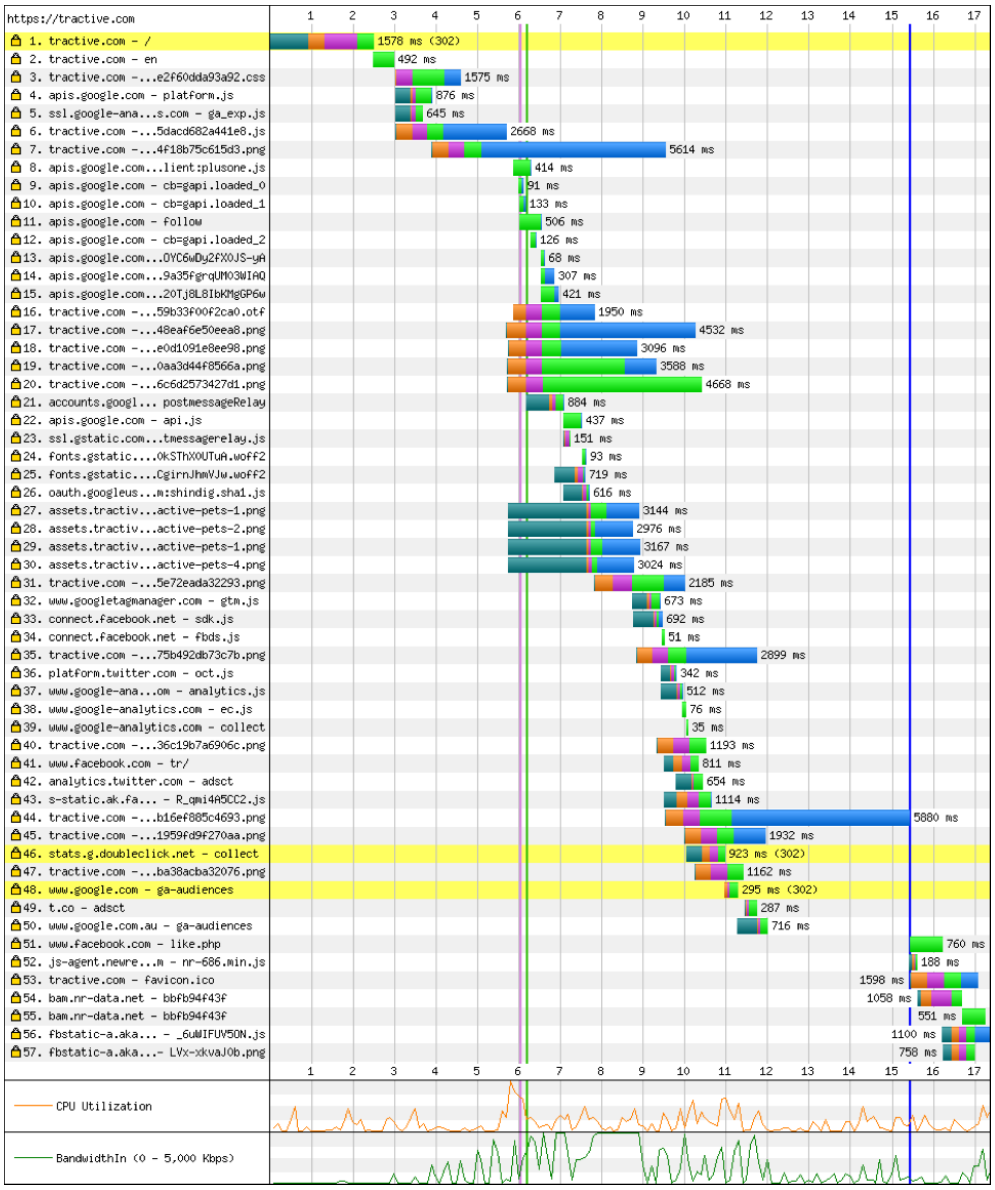

Figure 4.9: Waterfall view of https://tractive.com - Request from Sydney, Australia

just an indicator from webpagetest.org when the filmstrips taken are complete. More important are the columns **Document Complete** showing the number of requests, content downloaded and once again onLoad time respectively Fully Loaded including further requests after the document is loaded and rendered completely, as the favicon for instance.

The waterfall view shows each request with it's according time periods explained in the legend above. Already the first request introduces a time delay of 1578ms although it just retrieves a frew bytes. This heavily reflects our knowledge and results from the previous measurements because it is mainly latency which makes the difference here. Before any resources are downloaded a second request does a redirect to the desired language based on the Accept-Language: en-US,en;q=0.8 header from the first request.

This is a problematic request since it basically does nothing and costs nearly half a second. Finally the first byte has received at the browser and it now can start downloading necessary resources and building DOM and CSSOM. CSS and Javascript are critical resources because they block DOM creation and therefore delay the Start Render time. For formatting the page we have to load the application.css at the beginning followed by some Javascript files needed for Google Analytics and Google+ integration. They can be downloaded in parallel and does not introduce a big problem in this case because the application.css needs more time. A better option here is to defer loading of these Javascript files to a later time as we will discuss in 5 .However, the most critical asset here is the application.js file which contains a lot of custom Javascript functions including third party libraries like jQuery implementing client side logic and interactivity as well as AJAX and REST calls. It needs 2668ms to be downloaded and during this time the Javascript Engine is blocking DOM construction. During my further measurements I identified this asset as a potential bottleneck. Optimizing it can lower the Start Render time significantly and therefore makes the first elements of the page faster visible. Subsequently the rest of the DOM can be constructed and the green vertical line shows when the rendering of the page begins. Now all images can be downloaded and placed correctly. As expected the header takes very long to be downloaded since it is one of the biggest assets. Luckily it starts downloading pretty early to be finished in time because it represents the company and is an essential part of the main webpage. A little surprising is that the wooden background of the website is delayed until the 35th request what is probably not an optimal order. A closer look at the graphs at the bottom of the figure shows the CPU utilization during page load. There can be different reasons for high CPU utilization but usually Javascript consumes CPU a lot due to executions of functions which do computations, loop operations with higher complexity etc. In this test it shows a similar behaviour as the loading and execution of application.js introduces a peak here. Concluding this test I identify the initial redirect and the Javascript execution as major bottlenecks and therefore I will look at this in more detail in chapter 5.

4.5 Utilization measurements

Sometimes server capacities can introduce a bottleneck when load is too huge or other resources affect the server. This sections gives a short overview on the utilization values I found out for our three multi purpose application servers. A very convenient command to get the most essential utilization metrics is the **top** command available in all Linux environments.

A first thing was to compare the three servers regarding their common metrics given by top:

```
tractive03
top - 13:17:18 up 215 days,  8:02,  1 user,  load average: 2.53, 2.84,
    2.77
Tasks: 222 total,   3 running, 219 sleeping,   0 stopped,   0 zombie
Cpu(s): 13.0 us,  1.7 sy,  0.0 ni, 84.7 id,  0.0 wa,  0.0 hi,  0.7 si,
    0.0 sKiB Mem:  32639696 total, 25521968 used,  7117728 free,       48
     buffers
KiB Swap: 16768892 total,  1242236 used, 15526656 free.  7597212 cached
    Mem

tractive04
top - 13:17:31 up 38 days,  2:47,  1 user,  load average: 1.60, 2.75,
    2.77
Tasks: 218 total,   2 running, 216 sleeping,   0 stopped,   0 zombie
Cpu(s): 23.2 us,  3.9 sy,  0.0 ni, 72.1 id,  0.0 wa,  0.0 hi,  0.7 si,
    0.0 s
KiB Mem:  49453844 total, 38325948 used, 11127896 free,       64 buffers
KiB Swap: 25149308 total,  1103168 used, 24046140 free.  9939060 cached
    Mem

tractive02
top - 13:17:24 up 466 days,  5:56,  1 user,  load average: 2.14, 1.79,
    1.66
Tasks: 244 total,   1 running, 243 sleeping,   0 stopped,   0 zombie
Cpu(s): 27.7 us,  1.2 sy,  0.0 ni, 70.8 id,  0.0 wa,  0.0 hi,  0.3 si,
    0.0 s
KiB Mem:  32808072 total, 28066204 used,  4741868 free,       68 buffers
KiB Swap:  8384444 total,  1734596 used,  6649848 free.  6363004 cached
    Mem
```

As expected the servers operate pretty equal and therefore no application server seems to suffer from any problem. tractive04 has with 48GB of RAM the highest capacity and uses the most memory with approximately 38GB. The average load is nearly the same on every server due to the load balancer components. The same applies for CPU utilization which emphasizes the fact that there is no strange problem on any server affecting the web performance. The load average displays three numbers whereby the first one is the average in the last minute, the second in 5 minutes and the third one spans over the last 15 minutes. The numbers indicate the average amount of processes which were waiting for the CPU. Due to the fact that these servers use 4 cores there is nearly no process which has to wait for the CPU. In addition to that it gives information about uptime, amount of tasks and following this overview a detailed list of processes and their CPU respectively memory consumption. During my investigation I did not find any processes with uncommon behaviour.

Chapter 5

Suggested Improvements

The previous chapter covered the measurements which were taken to determine the overall web performance of the system under test. Furthermore some problems and potential bottlenecks affecting performance were outlined. The following sections try to tackle those problems and suggest possible improvements.

5.1 Content Delivery Networks (CDN)

The measurements regarding time spent in the network confirmed that latency is a major issue and one of the most important metrics which drives loading speed of a website, engineers have to care about it. CDNs are a great opportunity to improve network latency by bringing the content closer to the clients. Since Tractive operates in over 80 countries of the world, clients necessarily experience different latencies because the website is mirrored on three hosts residing in one datacenter in Germany. A CDN offers the possibility to replicate the content across a set of geographically distributed servers instead of putting them only at a single location. Based on the latency (which is mostly driven by the distance) the client gets redirected to the nearest node to get the requested content which minimizes round-trip-times for TCP connections, SSL handshakes and so on [9]. These geographically distributed nodes, which are called edge-servers, cache the requested content after the first time. So if the desired content is already available at the closest edge server it can be directly delivered and no extra round trip to the origin server is necessary. A popular CDN is Amazon Cloud Front which allows to distribute content of the website globally in Amazon's Availability zones. Cloud Front integrates perfectly if the web content is hosted in an Amazon Web Services environment [6].

CDNs can be used to deliver the entire website but also for single assets like images, javascript etc. In our new website for instance we deliver different Javascript libraries we use in our web application via the CloudFare CDN instead of storing them at our origin server in Germany. This should help to improve the critical rendering path and force loading time of the entire website to decrease.

```
<script src="//cdnjs.cloudflare.com/ajax/libs/store.js/1.3.17/store.min.
    js" charset="utf-8"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/1.9.1/jquery.min.js
    " type="text/javascript"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/flexslider/2.1/jquery.
    flexslider.js" type="text/javascript"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/flickity/0.1.0/flickity.
    pkgd.min.js" type="text/javascript"></script>
```

Listing 5.1: Loading Javascript libraries from Cloudfare CDN

5.2 Loading of Assets and Caching

Images, nice layouts using CSS, fancy animations or jQuery AJAX calls to backend services, all of them are critical in the rendering path and have to be loaded and executed fast to avoid diminishing web performance. As the measurements in **??** showed the main application.js file is a potential bottleneck because it delays the start of rendering significantly especially in case of high latencies. Fortunately Caching supports an intelligent solution to prevent unnecessary loading of assets from the origin server each time. Especially de-facto static content like product or advertising images on the main website should be cached within the browser to enable good user experience at recurring page visits. Fortunately we have Nginx as powerful reverse proxy and intermediary web cache sitting in between the load balancer and the application server. Nginx offers a rich set of caching policies to avoid requests always forwarded to the application server and asking for a resource. In the current state caching for static assets is leveraged enough but Nginx allows fine-tuning configuration even for dynamic content. Revalidation proxy cache setting for instance allows to check if there were any changes to dynamic resource and even if the cache header has expired delivering cached content.

5.3 Compression

When content has to be downloaded, file size of course influences visualization performance even though parallel downloads are often no problem. To get fast download times assets should be compressed with GZip which is supported by all popular web server. Also Nginx supports gzip static compression.

5.4 Language and Redirection

Providing web content in different languages is a duty for Tractive to be competitive in the global market. However, this introduces some challenges because the recommended language first has to be detected correctly and the web application has to know which set of translations it should take. To solve this Tractive used a simple redirect to indicate the used language (for instance tractive.com/en) directly after the first request. The measurement results showed that this is problematic for the starting render time since it involves a new HTTP request and further round trips. There are several possible solutions to solve this issue. The new website uses a Javascript file to detect the browser's language and compares it with the valid languages Tractive supports. As fallback language English is used. The following code snippet shows a part of the lang.js which reduces the time spent in the network from about 140ms to 45ms for a request from Frankfurt, Germany.

```
var validLangs = ["en", "de", "es", "it", "fr", "sr", "ru", "bg", "hr",
    "cs"];

// Get the browser language
function getBrowserLang() {
    return navigator.language.split('-')[0];
 }
 // Return the browser language if supported, fallback to English
     otherwise
function getChosenLang() {
    var bl = getBrowserLang();
    return validLangs.indexOf(bl) < 0 ? validLangs[0] : bl;
}
// Get localized root URL
function getLangRoot(lang) {
    return "/" + lang;
}
```

Listing 5.2: Code snippet of lang.js

After executing the lang.js file the appropriate HTML file in the correct language can be generated and returned to the client. With a rails application running in the back-end there is an even better solution to set the correct language. Since Rails generates the outcoming HTML with it's template engine ERB which listens to a global variable ”locale”, we can set this variable in the main application controller based on the Accept Language HTTP header. This allows to remove the javascript file from above completely and saves time before constructing the DOM and CSSOM. The only thing the Rails application server has to do is to download the Accept Language Header which is just a few kilobytes. Especially for dynamic websites which need a rich backend anyway such a solution makes sense whereas static websites such as the new developed main website tractive.com don't use a Rails application behind it anymore. Instead HTML pages are generated for the languages with some middleware software.

Chapter 6

Conclusion and Future Work

The work in this thesis gave detail insights in the key factors driving the performance of a web application. The web, it's data and complexity has grown immensely and web engineers nowadays have to have a broad horizon of what's going on in the network as well as in the browser. Network Latency is the most problematic unit when sending requests and responses over HTTP since it scales with the distance and intermediate stations on the way. Carefully thinking about where to place web content plays a key role here since transfer speed has a hard limit with the speed of light and refractions of the medium. The measurements confirmed the presumption that the network is highly dependent on other participants, routers or current load and therefore often unreliable. A bunch of concurrent and sequential requests result in high standard deviation and outliers since request rate of the web server can't grow that high to handle all requests equally. The Apache Benchmark command line tool with it's simple extensibility via shell scripts was used to execute load tests as well as consistency of requests over different time periods. Measurements showed that load also varies whether done in the night or during the day. How important it is to understand DOM processing, CSS trees and the Javascript Engine was illustrated when measuring processing in the browser with the webpagetest.org tool from Google. The waterfall diagrams showed the importance of the start render time and outlined problems regarding blocking Javascript files and URL redirects. At last a few improvements to eliminate existing problems were suggested and illustrated with short code snippets to give an idea how existing problems in the system can be solved or alleviated.

Due to the reason that Web Performance has become a hot topic, research and development is evolving continuously and there are still enough possibilities to save tens of milliseconds from the start of a request until it's rendering in the browser. Especially HTTP and it's underlying TCP connection still offers scope for improvements. HTTP/2 was approved in 2015 and sounds promising to give major advantages in reducing latency. Although it's core concept remained the same, it handles data differently under the hood. The usage of TCP connections gets more efficiently with a completely new binary framing layer which allows multiplexing of requests and responses over a single connection. With stream prioritization and server push new cool features wait to be leveraged in web applications and open opportunities which did not exist before [4].

Bibliography

[1] Patrick Chang. Fundamentals of http.

[2] Ilya Grigorik.

[3] Ilya Grigorik. *High-Performance Browser Networking.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, Sep 2013.

[4] Ilya Grigorik. *HTTP/2: A New Excerpt from High Performance Browser Networking.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, May 2015.

[5] Patrick Killelea. *Web Performance Tuning*, volume 2. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, Mar 2002.

[6] Sajee Mathew. Overview of amazon web services, 2015.

[7] Daniel Menasce et al. Scaling for e-business. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*, pages 511–513. IEEE, 2000.

[8] New Relic.

[9] Harsha V. Madhyastha Rupa Krishnan et al. Imc-09. In *Moving Beyond End-to-End Path Information to Optimize CDN Performance*, 2009.