2017 IEEE 42nd Conference on Local Computer Networks

Identifying the Performance Impairment of HTTP

Jens Heuschkel
TK / TU Darmstadt,
Darmstadt, Germany
heuschkel@tk.tu-darmstadt.de

Jens Forstmann
TK / TU Darmstadt,
Darmstadt, Germany
fostmann@tk.tu-darmstadt.de

Lin Wang
TK / TU Darmstadt,
Darmstadt, Germany
wang@tk.tu-darmstadt.de

Max Mühlhäuser
TK / TU Darmstadt,
Darmstadt, Germany
max@tk.tu-darmstadt.de

***Abstract*—Online web services have been constantly developed and offered through the Internet. These services rely on webpages that are hosted on remote servers and are transmitted to clients via HTTP upon requests. While HTTP has been updated to support more and more sophisticated services with increasing amount of multimedia content, the real-world adoption of those updates is still very slow. In this paper, we investigate the performance impairment in current HTTP implementations and suggest solutions to improve the situation. Specifically, we identify two insights, i.e., avoiding unnecessary DNS lookups by replacing domain names with prefetched IP addresses for linked resources and reducing TCP connections by attaching linked resources to HTML files as binary stream, that could be employed to largely improve the performance of web services without modifying HTTP itself. Through extensive measurements, we validate our insights and the results confirm our findings.**

***Index Terms*—Web services, HTTP, performance, measurement**

I. Introduction

Communication, education, entertainment, and searching, this list names only a few of the endless possibilities which are done online everyday. As a result, the Internet is clearly an important part of our daily life by now. Business applications or services are developed more and more as browser versions using HTTP (hypertext transport protocol) and most of our online interactions are carried out over HTTP nowadays.

HTTP was invented to transfer data without holding a state. As commonly known, it is mostly used to access webpages within the world wide web. HTTP was brought up in version 1.0 in the year 1996 [1] and was then updated to version 1.1 in 1999 [2]. Further RFCs have also been published to clarify the HTTP v1.1 specification. However, these protocols were not intentionally designed to serve today's heavy demands for web communication as webpages have been evolved to rich multimedia pages holding many pictures, videos, and interactive content. To handle the situation, HTTP recently got an update to version 2.0 in the year 2015 [3]. While many promising changes are introduced to improve the performance, the update is just rarely adopted.

On the other hand, various optimization strategies have been incorporated into browsers over time to speed up web browsing. For instance, browsers can request linked resources in parallel or cache them for later use to shorten the webpage loading time. On the other hand, HTTP updates have been adopted very slowly in real-world implementations, ignoring those new features that were introduced for a good reason. For instance, some browsers do not reuse open TCP connections. Reusing TCP connections could spare server resources but it seems to have a low benefit for the browser, since it introduces management overhead and resources are spread over many different servers for load balancing.

A service provider offering an online service via HTTP is interested in lowe page loading time to achieve quick response time. This will bring increased user experience as well as improved resource utilization of servers. Furthermore, fast webpage loading enables the possibility of offering services via mobile connections. Considering that mobile devices are becoming ubiquitous (and somewhat dominant), this will further lead to the opening of a huge business market.

In this paper, we investigate the performance issue in current HTTP implementations and suggest solutions on how we could speed up browser-server communication within the most popular HTTP v1.1 capabilities. In particular, we identify two insights that could be employed to largely improve the performance of HTTP-based web services: 1) avoiding unnecessary DNS lookups by replacing domain names with prefetched IP addresses for linked resources, and 2) reducing TCP connections by attaching linked resources to HTML files as binary stream. We carry out extensive measurements to validate our insights and the results confirm our findings. In particular, we made the following three contributions:

1) We investigate the main cause of performance impairments in the current practice in the web. 2) We present two insights on how to improve the HTTP connection without changing the protocol itself. 3) We carry out extensive measurements to evaluate our ideas and discuss the results.

The remainder of this paper is structured as follows: Section II presents our view on the problems and identify two insights on speeding up the web communication. Then, in Section III we present the evaluation of each of the insights, by showing our measurements for different network environments and discuss the outcome. Section IV shows the related work. Finally, Section V concludes the paper.

II. Methods

In this section we will discuss our view on the problems with current practice in web communication. Afterwards, we present our ideas how this could be solved.

A. Problem Investigation: Taking a deeper look into how requests are performed on the client's side offers some possibilities to maybe improve page loading times. Performing an HTTP request consists of many intermedi-

DOI 10.1109/LCN.2017.120

ate and partially independent steps. Assume a client loads `http://www.example.com/document.html`. A sequence typically looks like the following (Figure 1).

1) The browser requests the IP address for `www.example.com` from the operating system. The operating system looks up the IP address in its local cache. If no entry in its cache is available it delegates the request to the first DNS server it is aware of. Usually not only one but multiple DNS servers will be asked to get an appropriate answer. Once the operating system gets the corresponding IP address it tells the browser. 2) The browser initiates a TCP connection to the web server. 3) The client sends the HTTP request for `document.html`. The server responds with the content of the file. 4) The client parses the file after it was received completely. 5) Since an HTML document usually depends on other files such as stylesheets, javascripts, images and others which are linked within the document the client has to request them, too. To this end, a DNS lookup is performed for each of the linked files. 6) The browser initiates a TCP connection to each of the servers that contain the linked files. 7) For each linked resource the client performs a HTTP request. 8) As soon as all resources are received the browser renders the complete website.

Requests are sent to a specific server identified by its IP address, but before the browser knows the actual IP address it has to ask one or even several DNS servers to resolve the domain name. This step could take quite a bit time, depending on many factors. Only after this step is completed, the HTTP server can be asked for the requested resource. For every resource itself, a separate request is needed, and since resources are often located in content delivery networks, DNS requests have to be performed. As a consequence, it's necessary to perform hundreds of requests and thus to have possibly hundreds of DNS lookups and TCP connections. For instance, we observed 171 requests from 11 different servers with around 5 MB of data for loading the Amazon landing page. Grigorik et al. talk about an average of 90 resources, fetched from more than 15 distinct hosts with around 1300 KB of data [4]. Obviously, this process is getting quite costly and time-consuming. Hence, searching for any savings is expected. The good news is, browser makers already put significant effort into optimizing this process and there are some working improvements. In addition, HTTP got various improvements over time, which we will summarize in Section IV. Despite all these improvements, the time for requesting DNS is still a big issue. Moreover, the need for connecting to many servers for the resources, which results again in many DNS requests and in many separate TCP connections brings a big overhead and is not solved yet. This paper aims to contribute new ideas to further improve the performance of HTTP-based web services.

B. Our Insights: In current networks, latency is the most important factor for the user experience while browsing the web [5]. As a consequence, even small improvements on repetitive, latency affecting parts could achieve a big gain. Specifically, we present the following two insights.

Insight 1: Avoiding DNS lookups. Avoiding the DNS resolution would be a logical step to speed up the process. However, the domain names exist for a good reason, since IP addresses are not very convenient. Additionally, the IP addresses are not static and domain resolution can be used to balance the load on several servers. But for linked resources, nobody would bother if these resource are referenced via an IP address. At some point, the domain has to be replaces with a prefetched but load balanced IP address. Ideally, this would happen inside data centers where the needed information is available.

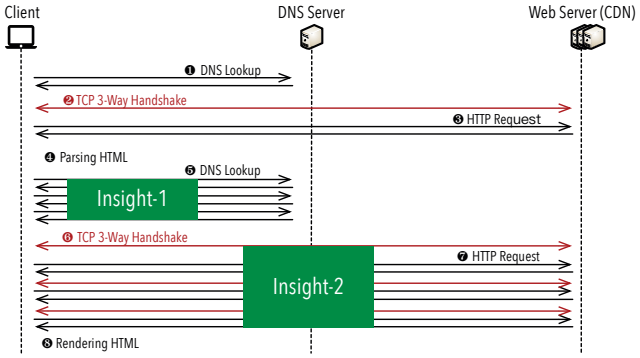

Fig. 1: The workflow of a typical HTTP web service and the parts where we identify the insights for performance improvements.

Insight 2: Reducing TCP connections. Another idea could be to reduce the number of different requests as well as of web servers. This would reduce not only the DNS request, but also the needed TCP connections where each need an additional TCP handshake. Especially, scripts and stylesheets can be easily embedded into the HTML file, eliminating the need for additional requests. However, this would reduce the maintainability of the page since developers have to make changes on many different places. Also, it would not be able to reuse the scripts or stylesheets for other pages, bringing a dramatic increase of network traffic. Compared to embedding, attaching those resources to the HTML file as binary stream or as linked resources has many benefits. The browser could handle them like separate loaded resources, leveraging all the well known and implemented improvements like caching.

III. Evaluation

This section presents the evaluation of our two presented insights. We made our scripts and measurement raw values publicly available at https://github.com/tklab-tud/FastHTTP. The measurement results are reproducible with the described toolset and settings described in this section.

A. Test Environment: To test under consistent conditions, we decide to use two virtual machines running on the same host. The exact system specifications are summarized in **??**. The hypervisor on the host machine is Oracle VM VirtualBox.

1) Web server VM: On the web server VM an Apache HTTP Server with its standard configuration Furthermore, a simple Dnsmasq provides the DNS resolution system. All web domains point to the web server VM which has eleven IP addresses in total.

2) Client VM: For testing the page loading times under different circumstances, a Mozilla Firefox is installed on the client VM. To determine how long it takes for each step (such as establishing a connection, looking up the IP address, requesting the resource and transferring the data) the timings API provided by the browser is used. The API can be accessed by JavaScript code embedded directly in the page itself or by any other JS code injected by the browser. For our evaluation we used a script which was injected by the Firefox extension Greasemonkey. After all resources are loaded the script requests the timings from the API and merges them together with some meta data into a JSON object.

B. Test Parameters: Changing the network parameters to simulate different environments is crucial to rate the result for each test case. We decide to investigate five different situations: 1) *LAN* as enterprise network, 2) *DSL* as a typical available home internet connection, 3) *Cable* as a fast available home internet connection, 4) *LTE* as a fast mobile internet connection within cities, 5) *3G* as a typical available mobile internet connection outside of cities. The parameters are listed in Table I (from the client's perspective).

preset name	download (Mb/s)	upload (Mb/s)	ping (ms)
LAN	100	100	2
DSL	50	10	24
Cable	100	2.5	30
LTE	75	35	70
3G	7	0.75	150

TABLE I: Network environment presets

To enshure the bandwidth limitations, traffic control (TC) is used. TC is a command line tool for Linux to manipulate traffic control settings. For our evaluation different queueing disciplines (qdisc) are defined to limit every VM in terms of outgoing throughput and delay.

C. Test Cases: Six different test cases are investigated: 1) *Multiple domains*: The website resources are spread over multiple (10) servers. The resources are linked by domains. 2) *Multiple IPs*: The website resources are spread over multiple (10) servers. The resources are linked by IP addresses. 3) *Single domain*: The website resources are located on a single server. The resources are linked by a domain. 4) *Single IP*: The website resources are located on a single server. The resources are linked by a IP address. 5) *Attached archive*: The website resources are provided as compressed archive (deflate). The archive is attached as a binary blob to the index file. 6) *Linked archive*: The website resources are provided as compressed archive (deflate). The archive is linked by the same domain as the index file. So it is de facto as link by IP address since the IP address for the domain is well known.

For all test cases we used a synthetic website inspired by the characteristics of the Amazon and Facebook landing page. The page contains the minimal HTML elements and includes 170 images. The total size is roughly 5.2 MB.

This work only investigates page loading times and not the rendering in the browser. So neither packing, extracting nor displaying the content plays a role in this test run. In this paper we focus on the payload transmission.

D. Measurement Results: We measured every testcase in every presented parameter preset. This adds up to 30 different testcases in total. The multiple domain test case is good for comparison as it reflects the current situation in the world wide web. To get reliable results every test is executed five times. Both the minimum and the maximum result is ignored. An average page loading time is calculated over the remaining three values. Table II shows the average of page loading times.

3) LAN: The first network environment in this test series is the LAN preset, with both download and upload speed at 100 MBit/s and a low RTT with 2 ms. It takes an average of 1226 ms to download the test website with 170 linked images distributed over ten servers which are identified by their domain names. Replacing all domain names with their corresponding IP addresses speeds up the page loading to 1131 ms ($\tilde{8}$% speedup). The resources are requested in parallel. Hence, we just have to wait for one time the DNS request and not for the sum of all DNS requests. An faster speed was measured with all content packed in an archive. An attached archive directly after the HTML document of the root resource which was requested and a linked archive are more than 26% faster than the normal multiple domains transmission. The attached archive page is loaded within 901 ms (26% speedup), but the linked archive page with 803 ms (35% speedup) is even faster. Transferring the page content via one server only gives only a very tiny performance boost: Loading via only one domain takes 1221 ms whereas with avoiding the DNS lookup it takes 1207 ms.

4) DSL: The DSL preset with its 50 MBit/s download and 10 MBit/s upload speed and a ping with 24 ms shows a smaller deviation in-between the different test cases. It takes 1760 ms to load the page from ten servers with DNS lookups and 1617 ms (8% faster) without those name resolutions. The attached archive page is loaded after 1682 ms (4% speedup), whereas the linked archive test case is finished after 1542 ms (12% speedup). Loading the 170 linked resources from one and the same server gives only slightly different times. The one domain test case takes 1703 ms to load the page, this is only 3% faster; however, requesting the same content but directly via its IP address takes 1595 ms (9% speedup).

5) Cable internet: The cable internet comes with the same download speed as the LAN preset which is 100 MBit/s, but with an upload speed of 2.5 MBit/s and a ping around 30 ms. Loading the test page with the 170 linked images distributed among ten servers with different domain names takes 1722 ms. Having the same test case but requesting the images directly via IP ends up with a page loading time of 1557 ms. This is 10% faster by just not having DNS lookups. Similar to the results of the LAN environment, the archive approach gives a big performance boost. The page with the attached archive loads in 1113 ms (35% speedup) whereas the page with the linked archive is loaded after only 924 ms (46% speedup). The one server approach with DNS lookups takes 1610 ms in this scenario which is a 7% boost compared to the multiple domains case. Loading the same content directly via an IP address takes 1362 ms (21% speedup).

	multiple domains	multiple IPs	one domain	one IP	attached archive	linked archive
lan	1226	1131	1221	1207	901	803
dsl	1760	1617	1703	1595	1682	1542
cable	1722	1557	1610	1362	1113	924
lte	2047	1960	2753	2612	1697	1428
3g	6562	5854	8581	8239	8361	7886

TABLE II: Average of page loading times in ms

6) LTE: LTE has a download speed of 75 MBit/s and an upload speed of 35 MBit/s. This throughput preset is a little bit faster than the DSL parameters, but with a high RTT of 70 ms. To load the test page with 170 linked resources from ten different servers identified by their domain names requires 2047 ms. Loading the same content from the servers but accessing them via their IP addresses takes 1960 ms (4% faster). For the attached archive case it takes 1697 ms to load the content (about 17% faster). Not attaching the content to the root resource but linking it speeds up the process even more to 1428 ms (30% faster). Both the one domain and the one IP address approach take longer with LTE compared to the multiple domains and multiple IPs test cases. It takes 2753 ms to load the page via a single domain name and 2612 ms with directly connecting to the IP address. This is 34% and 28% slower than the distributed test case, respectively.

7) 3G: The last test parameters are 7 MBit/s download speed and 750 KBit/s upload speed with a ping of 150 ms for the 3G connection. Loading all linked resources from all different servers by their domains takes 6562 ms for the 3G settings. It is 11% faster without DNS lookups (5854 ms). To receive the content within an attached archive it takes 8361 ms (27% slower). If the archive is not attached but linked in the root resource it is slightly faster, but still slower than the multiple domains test case: with 7886 ms it is 20% slower compared to the latter one. Receiving the 170 images from only one server identified by its domain name takes 8581 ms (31% slower). Skipping the DNS lookup for the one server test case gives a result of 8239 ms which is 26% slower.

In summary, it seems like our idea to replace domains with IP addresses is a very simple but powerful method to reduce loading times of web pages and web applications. On low latency links, it seems beneficial to load resources from a single server instead of many different servers. But this would bring up the need for other or new load balancing strategies. On high latency links, it is counterproductive to use just a single server. Except for the 3G scenario, it is very beneficial to create a archive with all needed resources and link them to the HTML document. This is not practicable for dynamic pages like Facebook, Twitter and co, but may help smaller pages with more static content like typical Wordpress installations.

IV. Related Work

There are a lot of existing solutions, to speed up the web browsing performance. We state the two most recent once here.

QUIC (Quick UDP Internet Connections) is a transport layer network protocol designed by Google to work together with HTTP2 to speed up HTTP connections[6]. According to tight HTTP2 connection it offers similar features like the multi-stream support. It is also designed to ensure low latency by avoiding handshakes and include encryption functions to avoid additional TLS handshakes. Since its first release it was also focus of researches [7]

HTTP/2 is an advancement of the old HTTP/1.1 version. It is based on SPDY, an experimental protocol developed by Google. In the new version, many new features are available like multiple streams, binary framing or reduction of overhead. A very important feature is, that it eliminates head-of-line blocking, but only on the application layer [4]. Since 2015 HTTP/2 is standardized and published [8]. Modern browser already support HTTP2, but it is rarely used by now.

V. Conclusion

In this paper we presented our ideas to optimize the current state of HTTP communication. Our approaches are realizable with no or very few modifications of the web browsers or servers. In our evaluation, we could confirm, that DNS still has a huge impact on the performance. Furthermore, we showed that our approaches give a performance boost up to 46%. On the other hand, our approaches are no silver bullet since the system has to decide for an solution with respect to the used communication technology.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of the project B2 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, "Hypertext transfer protocol – http/1.0," Internet Requests for Comments, RFC Editor, RFC 1945, May 1996, http://www.rfc-editor.org/rfc/rfc1945.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1945.txt

[2] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," Internet Requests for Comments, RFC Editor, RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2616.txt

[3] R. Peon and H. Ruellan, "Hpack: Header compression for http/2," Internet Requests for Comments, RFC Editor, RFC 7541, May 2015.

[4] I. Grigorik, "Making the web faster with http 2.0," *Communications of the ACM*, vol. 56, no. 12, pp. 42–49, 2013.

[5] ——, *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.", 2013.

[6] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "Quic: A udp-based secure and reliable transport for http/2," *IETF, draft-tsvwg-quic-protocol-02*, 2016.

[7] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 609–614.

[8] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," Internet Requests for Comments, RFC Editor, RFC 7540, May 2015, http://www.rfc-editor.org/rfc/rfc7540.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7540.txt