Concept for Client-initiated Selection of Cloud Instances for Improving QoE of Distributed Cloud Services

Florian Wamser, Michael Seufert, Steffen Höfner, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science, Würzburg, Germany
{wamser | seufert | hoefner | trangia}@informatik.uni-wuerzburg.de

ABSTRACT

We introduce a concept for client-initiated selection of service location and service quality for improving the Quality of Experience (QoE) of general cloud services. It is loosely based on the HTTP adaptive streaming approach (e.g., MPEG DASH). A manifest file compiled by the cloud service provider specifies the available service locations and qualities, from which the user selects the optimal service instance based on contextual information obtained from client measurements and user preferences. The proposed concept is defined and is implemented in two client-based decision algorithms for improving the QoE of a simple picture gallery cloud service. These decision algorithms are evaluated and their impact on the service delivery is discussed. The evaluation shows that it is possible to improve the service location and quality selection by light-weight client-based algorithms.

1. INTRODUCTION

Cloud services are growing at a fast pace and are ex- pected to grow to a $176 billion market in 2015 [1, 2]. This is due to the fact that the cloud offers the promise of almost unlimited and scalable resources. Thus, more and more application providers adopt their applications and move their solution to the cloud. Typically, cloud services are scaled in two dimensions, which influence the service delivery. First, there is the possibility to horizontally enlarge the service with multiple instances running simultaneously in different locations (virtual machines, servers, or data centers). Second, the cloud service can offer different service qualities. This vertical scaling of the service quality can be achieved by providing different functionality, delivering content in diverse qualities, or offering additional or reduced sets of features. For example, a video streaming service can be scaled horizontally by employing a content delivery network (service location) and vertically by providing different video bit rates and resolutions (service quality).

The decision on which location and which service quality shall be provided to the end user is usually taken on the server side based on simple heuristics, which are expected to reach a high Quality of Experience (QoE) for the end user. For example, a proxy service in the cloud or a DNS service can resolve the request to a particular cloud instance (service location) that is best suited according to the current workload in the data center. Another example is a web server, which delivers the mobile version of a website (service quality) based on the HTTP User-Agent header. However, taking this decision on the server side is not always the ideal solution as different users have different requirements regarding the location or service quality, which the server cannot be sure about. Mobile Internet users, for example, are mostly limited to a small screen size and are likely to have an unstable or low Internet connection. Thus, for a given service, a mobile user might prefer local servers to receive application feedback as quickly as possible to have a high QoE, and accept reductions in terms of content quality or functionality. PC users, on the other hand, have access to large screens with a high resolution. To have the best experience with the service, a PC user with a huge bandwidth might require the highest possible service quality instead of a close server location. As the servers usually have no knowledge of the user's context, e.g., device, connection, preferences, or expectations, adverse decisions might be taken.

A lot of this contextual information is present at the user's device, which is not and should not be communicated over the Internet, as long as the connection is not encrypted. Nevertheless, this information could be considered if the user (or the user's device) decides which service location and quality is to be accessed. Thus,

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in:

Internet-QoE, August 22-26 2016, Florianopolis , Brazil

c 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4425-8

DOI: http://dx.doi.org/10.1145/2940136.2940143

in this work, we propose a concept for a client-based selection for the improvement of QoE, which is loosely based on the HTTP adaptive streaming approach (e.g., MPEG DASH)[1]. When the user starts a specific cloud service, the client accesses a manifest file, which is compiled and provided by the cloud service provider and lists information about the available service locations and quality. Based on contextual information obtained from client-based measurements and user preferences, the optimal request can be sent by the client.

We show an implementation of two client-based selection algorithms for a simple picture gallery cloud service and discuss the impact on the service delivery. Thus, our work relates to cloud service providers, which target improved service delivery while achieving a high end user satisfaction. Therefore, the remainder of the paper is structured as follows. Background information and related work are presented in Section 2. In Section 3, the proposed concept is described and defined. Section 4 presents the experimental setup of the picture gallery cloud service, and Section 5 introduces the implemented client-based selection algorithms. In Section 6, the results of the evaluation are discussed, and conclusions are drawn in Section 7.

2. BACKGROUND INFORMATION AND RELATED WORK

We provide an overview of related work according to different directions, from today's basic mechanisms for distributed services to relevant concepts and ideas for specific applications on server and client side.

A fundamental way to improve the quality in today's cloud services is the distribution of content on different servers in different regions organized through content delivery networks (CDN). This principle is also used in the horizontal scaling for distributed cloud services where stateless or stateful cloud service instances are distributed across multiple servers in physically separated server farms. A critical task for the service quality here is the intelligent assignment of clients to servers: A common approach is the assignment according to the geographical distance by the server. Other approaches are DNS-based approaches on the server side [3,4]. In [3], the working principle of the Akamai CDN is explained, while [4] unveils Akamai's content delivery network and delivery technologies.

Related to our approach are works that deal with a client-side content server selection in CDNs [5–7]. [5] deals with the server selection process in the YouTube CDN, which is, among other factors, mainly based on round trip time between users and data centers. In [6] a client-side process is proposed that shares load-balancing functionality with CDNs by choosing from resolved CDN servers based on last mile network performance.

[1]In fact, our concept is a generalization of HTTP adaptive streaming to distributed and general cloud services.

There are several approaches that use machine learning for content selection for cloud services [8,9]. Learning provides a way to classify the existing parameters and to detect the most important decision parameters.

Recently, application-specific approaches are getting more attention. The *MPEG DASH Industry Forum* is working on implementation of adaptive video streaming with H.264/AVC and H.265/HEVC [10]. The content is divided into chunks and can be distributed to different servers so that the client can decide which quality it requests based on this current network conditoins. Comparisons of different streaming solutions can be found in [11,12] and the impact on QoE is discussed in [13,14]. The *Responsive Images Community Group* has standardized in HTML5 an approach to provide images in multiple resolutions for a web service [15,16]. The optimal image can be selected according to the client's viewport size. Another approach is to rely on third party solutions, which deliver images from the cloud, like *ReSRC.it* [17]. These services detect the screen size of the client and automatically resize the requested image to the optimal resolution before it is delivered for a cloud service.

3. CLIENT-INITIATED SELECTION OF SERVICE LOCATION AND QUALITY

The goal of the cloud service provider is to deliver the service to the user. From the perspective of the provider, the objective here is always to maximize the satisfaction of the user with the service while minimizing resource utilization on delivery. In order to meet both objectives, we define our concept with the premise that the client should access the best service instance out of a given number of service locations and qualities. **Our conceptual idea is based on the observation that significant information is available at the client-side, which the server is unaware of, and thus, will result in inferior decisions**. This includes information, such as user preferences, hardware capabilities, or the current Internet connection to the cloud service. The concept envisages to use this information for an optimal decision by the client on which service instance should be requested. From a QoE perspective, the information should be chosen so that the QoE is, in the end, positively influenced. Matching information for specific services and applications may need to be found in the literature on QoE models, e.g. [18,19].

3.1 Terminology

To begin with, the used terms will be defined. A cloud service consists of several service locations, which can be virtual machines, servers, or data centers, and are typically distributed in one or more clouds or networks (edge cloud). Each service location can be uniquely identified by the corresponding address. Moreover, a cloud service offers different service qualities, for example, different functionality, content, or features. We de-

fine a *service instance* as a single combination of service location and quality. A *decision algorithm* is a client-side software that selects a service instance to send the service request to. Moreover, we define a *manifest* as a list of available service instances, which is compiled and provided by the cloud service provider, and from which the decision algorithm can select. Finally, we define *contextual information* as all information related to the user's service experience that can be measured by a software on the client-side. This includes user preferences and expectations, device capabilities, QoS parameters of the network connection, and application-level QoE parameters. Contextual information is represented as one or more metrics, which will be exposed to and taken into account by the decision algorithm(s).

3.2 The Process of Service Instance Selection

In the following, the process of service instance selection is described. The concept is visualized in Figure 1. Once the user accesses the cloud service, the client first accesses the manifest of the cloud service, which provides a list of available service qualities per server location. *Note that the cloud provider may influence the decision process by providing only a subset of service instances in the manifest depending on its own criteria, e.g., workload, network traffic, or costs.* Second, the client gathers contextual information and passes the corresponding metrics to the decision algorithm, or the decision algorithm conducts the measurement itself. Based on these metrics, the decision algorithm selects an optimal service instance and requests the corresponding service quality from the corresponding service location. Depending on the nature of the cloud service, the service instance selection might be repeated and different decisions might be taken over time. Thus, at any time the decision is made specifically according to the user's needs.

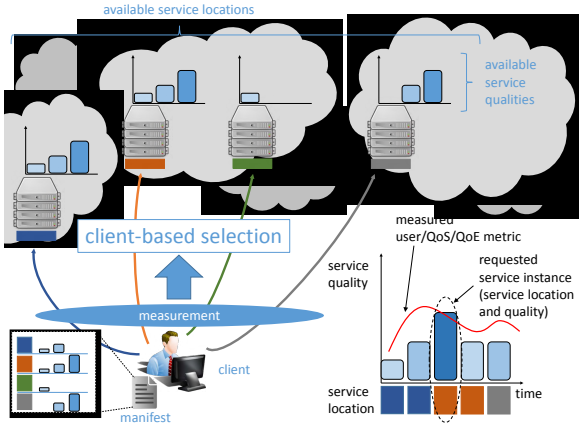

Figure 1: Visualization of the concept with multiple service locations and qualities and client-based selection of service instance according to different metrics.

4. EXPERIMENTAL SETUP

This section illustrates the experimental setup that was used during the study on the concept, to quantify the characteristic effects and benefits.

On the server side, we use Node.js with the express module as web server for the implementation of a simple picture gallery. Node.js is a JavaScript runtime which is built on Chrome's V8 JavaScript engine and is used to develop web applications. The picture gallery takes advantage of the cloud and distributes the picture content within multiple server instances in different qualities.

The hardware setup consists of three virtual servers at different locations and one local client. One of the three servers is running the web server which provides access to the picture gallery plus its content. The remaining two servers do nothing but provide the distributed content of the picture gallery. Moreover, the client is connected to the Internet via an OpenWrt router, which is used for emulating different network conditions.

As content for the testbed, we populated our picture gallery with 10 different high-resolution images. The original images were scaled down multiple times in order to create four different quality grades namely small, medium, large, and extra large. All of these scaled images have a JPEG quality factor of 100%. An overview of the different quality grades can be found in Table 1.

Table 1: Resolution and File Size of Images

Image Size	Small	Medium	Large	X-Large
Max. Width [px]	320	640	1024	2048
Avg. Size [MB]	0.06	0.17	0.41	1.52

5. SELECTION ALGORITHMS

In this section, we present different content and server selection algorithms, which can be executed on the client-side to improve the performance of the picture gallery service. First, we show an algorithm, which selects the content server with the lowest latency according to measurements conducted by the client. Then, we introduce an algorithm, which improves the Quality of Experience of the end user by selecting the image size, such that an acceptable picture loading time can be achieved under the current network conditions. Please note, that both algorithms can eventually be combined to improve the service by selecting the appropriate content and servers.

5.1 Algorithm 1: Selection of Content Server by Latency

The purpose of this algorithm is to select the optimal server. This can be useful, e.g., in a mobile networking scenario where the user prefers loading time over content quality. Therefore, it measures the latency time between the client and each server in the manifest file. Due to the lack of a `ping` function in JavaScript, it was necessary to develop a lightweight alternative. Therefore, an image file (`test1.bmp`) with a minimal

file size of 58 B and resolution of 1x1 pixel was placed on each content server. The algorithm downloads this image from each of the content servers and compares the download times. This process is repeated every 30 s to ensure the selection of the best content server also for changing conditions. This method of latency measurement is not as accurate as most other latency measurement functions, e.g., the built-in `ping` function of Windows or Linux. However, it is possible to roughly compare multiple servers to each other and to determine the best server among them, because the time delay that is caused by JavaScript and the short download should be similar for all measurements. The pseudo code can be seen in Algorithm 1.

```
while Latency Algorithm Active do
    forall the Content Servers do
        Download test1.bmp()
        Store Download Time()
    end
    Compare Download Times()
    Set New Best Content Server()
    Sleep(30 s)
end
```

Algorithm 1: Select Server by Latency

5.2 Algorithm 2: Selection of Content by QoE

Quality of Experience (QoE) of web-based services, such as picture galleries, is mainly influenced by waiting times [19]. Thus, this algorithm targets a high QoE of the end user by trading off content size and download times. The size of the requested image is maximized subject to the condition that it can be downloaded in an acceptable time. Therefore, the current throughput is estimated at the client and the maximum image size is computed taking the maximum picture loading time into account. According to [19], a good QoE (MOS > 3.5) can be achieved for picture loading times smaller than 1.3 s. Taking the average file sizes for each image size into account, we can calculate the required throughput to download a small, medium, large, or x-large image in less time. The pseudo code of the QoE-aware algorithm is presented in Algorithm 2.

In order to estimate the throughput of the client, two approaches will be utilized and compared. The first approach actively downloads a predefined image file with an exact file size of 1 MB from the current content server (*on idle*). After the download has finished, it calculates the average throughput by dividing the file size of the image by the measured download time. This measurement procedure is repeated every 30 s, if the user is not currently requesting another image. Note that this algorithm will increase the (mobile) traffic demand of the client. The second approach measures the available bandwidth passively whenever the user requests an image (*on the fly*) by reading the `Content-Length` field of the HTTP header to gather the size of the image.

```
while QoE-Aware Algorithm Active do
    Get Throughput()
    New Image Size = "x-large"
    if Throughput ≤ 9.84 Mbps then
        New Image Size = "large"
    end
    if Throughput ≤ 2.64 Mbps then
        New Image Size = "medium"
    end
    if Throughput ≤ 1.12 Mbps then
        New Image Size = "small"
    end
end
```

Algorithm 2: Select Image Size by Throughput

6. EXPERIMENTAL RESULTS

In this section, we carry out the evaluation of the concept. We first discuss the resulting complexity of the algorithms for the system. Then, we go into detail about the functioning of the individual algorithms and consider the accuracy of the selection algorithms and the achieved gain with respect to the goal of the algorithm.

6.1 Complexity of the Algorithms and Signaling Overhead

In this section, we briefly present our findings about the complexity of the algorithms. Since all algorithms are extremely simple and lightweight, they do neither influence the page load time nor the loading time of the content in a notable way. However, this only refers to the time complexity of the algorithms. The decision of the algorithms, e.g., the selected image size or the selected server, may indeed influence the loading time of the content, which is discussed in Section 6.

For the analysis of the signaling overhead, we neglect the initial download of the manifest file as these files are typically very small and are only downloaded once at the beginning. Moreover, we can neglect the algorithm, which measures the throughput on the fly. This algorithm does not create any additional traffic. However, the two remaining algorithms, i.e., latency measurement and throughput measurement on user idle, will constantly create additional traffic on the up- and downlink. In order to quantify this traffic, we performed 10 consecutive test runs. In each of these test runs, we recorded the communication overhead between client and server for the period of 30 min for each of the two algorithms. The results of this measurements can be found in Table 2.

Table 2: Average Overhead of the Different Algorithms and 95 % Confidence Intervals

Algorithm	Latency	Throughput on Idle
Uplink	0.267 ±0.001 Kbps	6.95 ± 0.29 Kbps
Downlink	0.271 ±0.002 Kbps	290.75 ± 2.53 Kbps

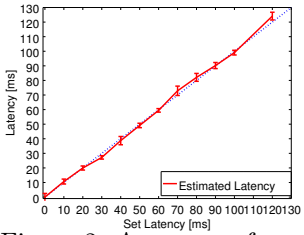

Figure 2: Accuracy of Latency Estimation.

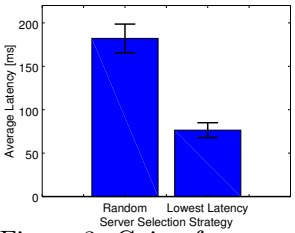

Figure 3: Gain of Algorithm 1.

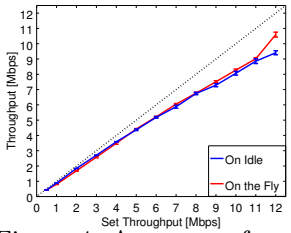

Figure 4: Accuracy of Throughput Estimation.

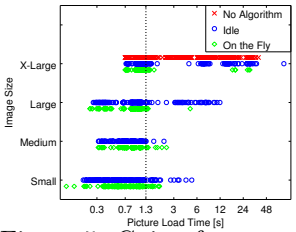

Figure 5: Gain of Algorithm 2.

The table shows that the latency estimation algorithm creates only very little traffic both on the uplink and the downlink. Still, the measured results are significantly higher than the theoretical overhead of 58 B of payload every 30 s (0.02 Kbps), mainly because of a comparatively large HTTP header of 615 B. Nevertheless, we conclude that it is suited for any type of Internet connection, even if the throughput is highly variable or very low. The throughput estimation, which downloads a 1 MB file every 30 s, creates a theoretical overhead of 279.62 Kbps. Here again, the measured downlink overhead is higher due to the overhead of the protocols. Although it will not negatively impact the Internet experience of users with a high bandwidth, this throughput measurement approach will for sure affect users with a low bandwidth, e.g., mobile users. Furthermore, the additional traffic could drain the data plan of mobile users, thus, the throughput estimation "on the fly" should be preferred in this scenario.

6.2 Algorithm 1: Selection of Content Server by Latency

The accuracy of Algorithm 1 depends on the accuracy of the incorporated latency measurement. We evaluate our approach by setting different latency values from 0 ms to 120 ms at the client using the popular NetEm tool. Figure 2 shows the set latency on the x-axis and the average estimated latency and 95 % confidence intervals of 50 measurements each in red on the y-axis. The baseline latency (i.e., measured latency to the server due to the local LAN without the use of NetEm) was subtracted from all measured values. It can be seen that for all set latencies the estimated latencies are close to the bisector (dashed), which shows the accuracy and applicability of our approach. The confidence intervals overlap the set values, while having acceptable sizes.

In the following, the latency-based server selection is compared to a random server selection in terms of resulting latency. Therefore, the latency of three servers (one in Germany, two in the United States) has been measured every 30 s for a total test run duration of 85 min. The test run was executed over the Internet with fluctuating network conditions and average latencies of 112.68 ms (median 55 ms), 185.92 ms (median 155 ms), and 247.79 ms (median 217 ms), respectively, were observed. Figure 3 shows the resulting average latency and the 95 % confidence intervals based on the different server selection strategies. The random server selection would randomly select one of the three servers to forward the client's requests, which results in an average latency of 182.1 ms. The lowest latency strategy of Algorithm 1, which selects the server with the lowest latency based on the latency estimation, can significantly reduce the average latency to 76.5 ms.

6.3 Algorithm 2: Selection of Content by QoE

Finally, we investigate Algorithm 2. As it is depending on an accurate throughput estimation, we evaluate the two estimation methods by setting different throughput limitation values from 0.5 Mbps to 12 Mbps for 100 requests each at the client using NetEm. Figure 4 depicts the average and 95 % confidence intervals of the throughput estimation for different set throughput thresholds. It can be seen that both the "on idle" (blue) and the "on the fly" (red) algorithm have similar estimations, which are close to the bisector (dashed) for low throughput limits. For increasing throughput values, the estimated throughputs deviate from the bisector by having smaller values than the set throughput.

To evaluate the gain of the QoE-aware content selection, we started 100 runs of 10 consecutive image requests with an interval of 30 s between each run. The throughput limit was rotated every 120 s among the following values: {0.5, 1, 2, 4, 15} Mbps. Figure 5 shows the picture loading times on the x-axis and the requested image size on the y-axis. A vertical dashed line indicates the critical picture loading time of 1.3 s, which results in a just good MOS value of 3.5 [19]. The baseline scenario without any algorithm, which always downloads the x-large image size, is plotted in red. It can be seen that only 40.2 % of the requested images have a loading time of less than 1.3 s, and thus, a good QoE in case of the baseline scenario. In contrast, the QoE-aware algorithms are able to increase the percentage of good QoE to 82.3 % (on idle) and 93.7 % (on the fly), respectively. However, a trade-off with respect to the image size has to be taken into account. The worse performance of the "on idle" algorithm can again be explained by the fact that it only updates its estimate every 30 s if the user is not interfering with a content request. Thus, this algorithm has to measure the through-

put within the 30 s idling window in between each test run. If it fails to catch this time window, the whole next test run will have an old, outdated throughput estimation in case of a throughput change. In contrast to this, the "on the fly" algorithm will only fail to request the correct image size once per throughput change.

7. CONCLUSION AND OUTLOOK

The goal of this paper was to develop and design a concept of client-initiated service location and quality selection for distributed cloud services. The concept was defined and evaluated in a proof-of-concept implementation of a cloud service picture gallery. Therefore, we developed one algorithm for quality grade selection and one algorithm for server selection based on client information, e.g., the client's network status.

The goal of the quality selection algorithm for the picture gallery is to deliver the requested content in an acceptable time to improve QoE. For this purpose, it measures the current available bandwidth and selects an appropriate image resource. The algorithm can use different throughput measurement strategies. One strategy is to periodically measure the available bandwidth by downloading a specific image file with a known file size. However, this approach causes additional network traffic, which will quickly drain the data volume of mobile users. The second approach is to measure the download speed on the fly, whenever the user requests and downloads an image resource. Therefore, the algorithm uses information about the image's file size in the HTTP header. Our results show, that the *on the fly* approach clearly outperforms the periodic measurements in our testing scenario.

For the server selection, we developed a simple and lightweight algorithm, which periodically measures the latency between the client and each content server that is specified in a manifest file.

Overall, we showed in this work that it is beneficial for web services to make use of algorithms, which select the server and quality of embedded images based on client side parameters in order to improve loading times. For future work, we plan to improve our content selection algorithms and to examine the use of client-sided selection algorithms for other cloud services and content types, e.g., documents, audio, or even gaming.

8. ACKNOWLEDGEMENTS

This work was supported by the H2020 INPUT (In-Network Programmability for Next Generation Personal Cloud Service Support) project and by Deutsche Forschungsgemeinschaft (DFG) under grant TR257/41.

9. REFERENCES

[1] Gartner, "Forecast Analysis: Public Cloud Services, Worldwide, 1Q15," Whitepaper, 2015.

[2] M. Armbrust *et al.*, "A view of cloud computing," *Communications of the ACM*, pp. 50–58, 2010.

[3] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[4] J. Pan, Y. T. Hou, and B. Li, "An overview of dns-based server selections in content distribution networks," *Computer Networks*, vol. 43, no. 6, pp. 695–711, 2003.

[5] R. Torres *et al.*, "Dissecting video server selection strategies in the youtube cdn." in *ICDCS*, 2011, pp. 248–257.

[6] U. Goel, M. P. Wittie, and M. Steiner, "Faster web through client-assisted CDN server selection," in *ICCCN*, 2015.

[7] J. S. Otto *et al.*, "namehelp: intelligent client-side dns resolution." in *SIGCOMM*, 2012.

[8] P. Ahammad *et al.*, "A flexible platform for qoe-driven delivery of image-rich web applications," in *ICME*, 2015.

[9] M. Claeys *et al.*, "Design and optimisation of a (fa) q-learning-based http adaptive streaming client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014.

[10] DASH Industry Forum, "Guidelines for implementation: Dash-if interoperability points," http://dashif.org/wp-content/uploads/2015/10/DASH-IF-IOP-v3.1.pdf, 2015.

[11] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *MMSys*, 2011.

[12] T. Hoßfeld *et al.*, "Close to Optimum? User-centric Evaluation of Adaptation Logics for HTTP Adaptive Streaming," *PIK*, vol. 37, 2014.

[13] M. Seufert *et al.*, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, 2015.

[14] M. Jarschel *et al.*, "SDN-based application-aware networking on the example of youtube video streaming," in *European Workshop on Software Defined Networks (EWSDN)*, 2013, pp. 87–92.

[15] Responsive Images Community Group, "Responsive images," https://responsiveimages.org/.

[16] "W3c working draft 08 october 2015," http://www.w3.org/html/wg/drafts/html/master/semantics.html.

[17] "Resrc," http://www.resrc.it/, (Online: Accessed on 03. November 2015).

[18] F. Wamser, P. Casas, M. Seufert, C. Moldovan, P. Tran-Gia, and T. Hoßfeld, "Modeling the YouTube Stack: from Packets to Quality of Experience," *Computer Networks*, Mar. 2016.

[19] S. Egger *et al.*, "Waiting Times in Quality of Experience for Web Based Services," in *QoMEX*, 2012.