QoE Analysis of Spotify Audio Streaming and App Browsing

Anika Schwind, Lorenz Haberzettl, Florian Wamser, Tobias Hoßfeld
{anika.schwind|lorenz.haberzettl|florian.wamser|tobias.hossfeld}@informatik.uni-wuerzburg.de
University of Würzburg, Institute of Computer Science
Würzburg, Germany

ABSTRACT

Spotify is the most-listened audio streaming provider in 2019 with 217 million active users per month. Providers are therefore interested in the quality and functionality of Spotify in order to provide their users with the best possible streaming quality. While video streaming services such as Netflix and their streaming approach have been extensively explored in previous research, audio streaming services like Spotify and their corresponding behavior at certain network conditions have not been considered in detail yet. In this paper, we perform a QoE analysis under various network conditions and examine the app browsing performance of the audio streaming platform Spotify using its native Android mobile application. We have developed a measurement tool that emulates a user listening to audio through Spotify. While streaming, application and network layer parameters are captured that have a high correlation to the user's QoE. The paper shows a baseline scenario including the streaming of a single song as well as playlist streaming behavior. Next, the effect of interruptions on the streaming behavior is evaluated and finally, the influence of network impairments on QoE key performance indicators such as initial delay is shown.

ACM Reference Format:
Anika Schwind, Lorenz Haberzettl, Florian Wamser, Tobias Hoßfeld. 2019. QoE Analysis of Spotify Audio Streaming and App Browsing. In *4th Internet-QoE Workshop: QoE-based Analysis and Management of Data Communication Networks (Internet-QoE'19), October 21, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3349611.3355546

Internet-QoE'19, October 21, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6927-5/19/10…$15.00
https://doi.org/10.1145/3349611.3355546

1 INTRODUCTION

In the Android Google Play Store, the mobile Spotify application is one of the three most used applications worldwide[1]. A user spends about 25 hours a month on Spotify in case he or she belongs to one of the 100 million premium subscribers [1] among 217 million active users per month [5]. With regard to the global traffic mix on the Internet, Spotify alone accounts for remarkable 0.34 % of the total downlink Internet traffic [7]. Despite being one of the most popular apps, there is still little research that sheds light on Spotify's performance, the streaming behavior, and other key performance factors affecting the Quality of Experience (QoE).

Although the volume of data is not critical, functionality of Spotify in different situations is of high interest to network providers and operators. They seek for information about the buffering strategy and the need for Internet access when performing certain functions in order to optimize coverage and network deployment to satisfy users and guarantee good QoE. The streaming behavior when playing songs is different depending on the software platform, which also has been already studied in [9] for the Spotify web player in the browser. Overall, the connectivity of the Internet has a great impact on the availability of features in the application and the playback of a song from a playlist on Spotify. In the following, we give insight into our research on the behavior and operation of Spotify in terms of user-relevant parameters that have a high correlation with the actual QoE of a Spotify user.

The contribution of the work can be split into two main parts: First, we developed a testbed to determine the QoE of Spotify's mobile application measuring different key performance indicators (KPIs) for different network conditions using the subjective QoE studies of [6]. Second, the measurements allow extracting application and network layer data to identify minimum network requirements. By utilizing the findings of this work, ISPs can develop new techniques to improve their traffic and network management.

The remainder of this work is structured as follows. First, Section 2 provides background information and gives an overview of related work in this field. Next, Section 3 describes the methodology used in this work, including the

[1] https://play.google.com/store/apps/top, visited 2019-07.

setup of the testbed, the measuring procedure, and the monitored data. Afterward, Section 4 examines and interprets the results. Finally, Section 5 concludes this work by summarizing the findings and giving an outlook.

2 BACKGROUND AND RELATED WORK

Today, Spotify is one of the most important streaming provider as they offer applications to stream audio files on most modern devices, like Android and iOS Smartphones as well as desktop computers. Using their service, Spotify provide access to over 50 million tracks [4], which includes different songs, full albums, pre-generated and custom playlists, and podcasts.

When thinking about playing an audio file which has to be transmitted over the Internet, the trivial approach would be download-then-play. It requires the file to be transferred entirely before the playback can be started, which results in long waiting periods. Especially in mobile networks with poor network conditions, a single song can contain over 10 MB of data. In contrast to the aforementioned download-then-play, streaming allows the user to consume the media already when it has not been completely transferred yet. Playback can already start when a given amount of data is stored in the buffer. This buffer also smoothes playback by overcoming network specific factors like latency and packet loss which leads to a higher QoE.

Video streaming, especially YouTube, in conjunction with QoE has already been analyzed in-depth. In [11] the streaming behavior and the corresponding QoE of YouTube's native Android app is examined and compared to YouTube's mobile website. This approach utilizes a wrapper app, which gains information about the YouTube app by interacting with its GUI using UI Automator. In contrast to the passive measurements, [8] conducted large scale active measurements to quantify the impact of parameters from different layers on YouTube's QoE. [2] classifies the users' QoE when watching YouTube videos based on the monitored encrypted network traffic. In addition to YouTube, [3] also analyses the network characteristics of Netflix. In the above mentioned studies, objective KPIs were monitored. Those KPIs provide detailed information about the perceived video streaming quality and can be mapped to QoE by means of QoE models [10].

In contrast to video streaming, there is a lack of research for the QoE of audio streaming. Here, only few aspects were evaluated, yet. In [6], the authors compared the impact of temporal impairments between audio and video streaming by conducting a subjective user study. They found that music streaming users are less tolerant for both, initial delays as well as stalling. In addition, they presented two QoE models which allow to map measured initial delay and stalling values to mean opinion score (MOS) values. A first look into the

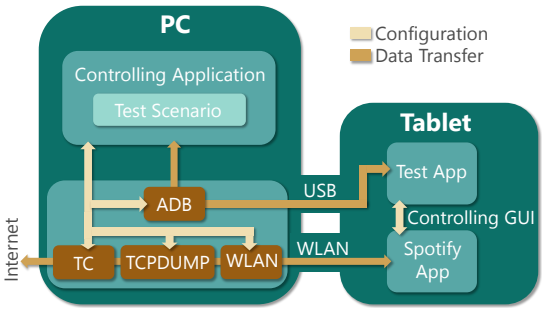

Figure 1: Testbed setup

streaming behavior of Spotify's web player was taken by [9]. Here, the authors presents a QoE measurement tool running inside a Docker container, which simulates a user listening to music. Using their tool, they monitored application layer as well as network layer KPIs while streaming, and mapped the measured values to the corresponding MOS.

Until now, authors heavily focused on video streaming and analyzes specifically on audio streaming are rare, especially for mobile applications. The opportunity of evaluating audio streaming and its KPIs in a controlled environment over a long period is missing. Thus, in this work, we present an active measurement testbed, which allows to conduct repeatable measurements under selected network conditions for Spotify's Android mobile Application.

3 METHODOLOGY

To characterize a Spotify audio streaming session using their mobile application and to quantify QoE related factors, a testbed was designed, which collects data on application as well as on network layer. Furthermore, a user is emulated, who streams audio in mobile networks using Spotify's mobile app to be able to replay user interaction. The design of the testbed was inspired by the testbed presented in [11]. As Spotify doesn't provide statistics about the playback like YouTube's "stats for nerds", other possibility to collect playback informations had to be used. Tests were done on an Android tablet, under various conditions, including user interaction and bandwidth limitation. In the following, the setup of the testbed is illustrated and the measurement procedure is explained. Afterwards, the monitored parameters on application and network layer are presented.

3.1 Testbed Setup

The structure is presented in Figure 1, which consists of two main components: The first one is a PC running Ubuntu 18.04 with the most current Linux Kernel available (version 4.20). The PC operates a NodeJS application (further referenced as controller), observing all tests and their components by performing particular test scenarios. The second one is a

tablet, the Acer ICONIA Tab 10, with Android 7.0 running the latest release of Spotify's mobile app, downloaded via the Google Play Store, which is currently version 8.4.88.150.

When starting a new test instance, the controller has to interact with the Spotify app on the tablet, instructing it to set specific options and start playback of a given song. As direct communication between them is not possible, a helper app is used. This app is an Android instrumented test, an app to test Android apps and their functionality. It runs in the background without any graphical user interface. Using UI Automator, it can interact with Spotify's GUI controls the same way as if a human person would manually conduct the test and press all the buttons and labels in the right order. Amongst others, UI Automator allows extracting relevant information from the GUI, like the current playback position in seconds, which is needed to calculate the initial delay and to detect stalling.

The Android Debug Bridge (adb) is used to deploy, to start and to stop the test app. In the testbed, adb communicates over USB. It also allows TCP port-forwarding over its USB connection, here this feature is used to forward port 8765 on the tablet to port 8765 on the PC, enabling communication between the controller and the test app. Adb's port forwarding is set up and managed by the controller.

To communicate with the Spotify's servers, a network connection is necessary. Therefore, the PC, which is connected to the Internet by a 250Mbps downstream and 40Mbps upstream connection, spins up a Wi-Fi network utilizing a wireless adapter (TP-LINK TL-WDN3200) attached by USB, bridged with its LAN interface allowing the tablet to access the Internet. Because of the Wi-Fi adapter's physical speed limitation, maximum throughput is limited to 24Mbps downstream and 22Mbps upstream. In order to simulate conditions prevailing in real mobile networks and to analyze the prerequisites of the Spotify Android app regarding the network, the network traffic has to be manipulated. This is the reason why adb is used to forward port 8765 onto which the controller and the test app communicate. For shaping the traffic flowing through the wireless network, the PC employs Traffic Control (tc) in conjunction with Network Emulator (netem). This way the controller can manually apply specific rules, affecting only the incoming traffic, like limiting the bandwidth, adding latency and jitter. While UI Automator allows extracting application layer data, in addition, the network layer is monitored using tcpdump.

3.2 Measuring Procedure

First, the test app is deployed and started. Afterwards the controller is started which handles the port forwarding, establishes a socket connection with the test app, starts impairments on the Wi-Fi network and sends the test configuration to the tablet. After receiving instructions for a new test, the test app prepares the device. This preparation includes closing all open applications as well as force-stop Spotify and deleting all data concerning the app to ensure that no information about the previous session remains, which could distort the results. The test app now informs the controller to start tcpdump.

To start the streaming, the Spotify app is opened and a test user with premium account is logged in to the app. Afterwards, the audio quality is entered, autoplay is disabled, and all available playlists are deleted. For each measurement run, a new playlist is created in which all songs concerning the current test are added. Next, the first song in the playlist is clicked to start playback and the playback view, where the title's cover and the current song position are visible, is opened up. Here, application layer parameters are monitored and manual interaction, like pressing the pause button, are emulated. When all songs have been played, the pause button transforms into a play button to repeat the playback, the test has been finished and all gathered information is sent back to the controller which stops collecting network information.

For the baseline measurements, we first used a playlist with one single song[2]. Additionally, we also conducted measurement runs focusing on the streaming behavior of a playlist with five songs. Here, in addition to the first song, four other songs[3] are added to the playlist. Finally, we added different bandwidth limitations to evaluate their influence on the streaming behavior and thus, the perceived QoE of the end users.

3.3 Monitored Data

Each test has been conducted between 23 and 27 times. For each measurement run, QoE KPIs on network as well as on application layer are stored.

On application layer, the current playtime is logged during the streaming of the audio file. In addition, the initial delay, i.e. the time between the request of the audio playback (click on the play button) and the start of the playback, is monitored. Besides the streaming parameters, also all page load times are logged from the login page of Spotify until the end of the audio playback. The aggregation of all these loading times from the start of the app until the start of the audio playback is herein after referred as navigation time. This includes starting and logging into the app, set settings, manage playlists, and search for songs. In addition, a network dump is created which can be used, for example, to calculate

[2] "Diamond Heart" by Alan Walker (3:59 min)

[3] "Dreamer" by Axwell and Ingrosso (4:11 min), "Bad Moon Rising" by Creedence Clearwater Revival (2:21 min), "Melody" by Lost Frequencies and James Blunt (2:29 min), and "Jennie" by Felix Jaehn, R. City, and Bori (3:04 min)

the packet arrival times and thus, calculate the download strategy of the audio file.

4 RESULTS

In this section, the evaluation of the measurement results are discussed. First, the baseline scenario is investigated, including the streaming of one single song as well as the streaming behavior of playlists. Next, the effect of interruptions on the streaming behavior is evaluated and finally, the influence of network impairments on QoE KPIs like initial delay is shown.

4.1 Baseline Scenario

To investigate the streaming behavior of Spotify's mobile application under optimum conditions, we conducted baseline measurements without limited bandwidth or user interaction. Because of the Wi-Fi adapters physical speed limitation, the bandwidth was up to 24Mbps downstream and 22Mbps upstream. Here, the median initial delay was only 1.06s and no stalling occurs. Furthermore, the median of the navigation time (time between starting the app and the actual start of the audio playback) was 53.12s. The song was streamed with a bitrate of about 96 kbps in the Ogg Vorbis format.

Looking at streaming only one single song, we found that Spotify's streaming policy tries to download the entire song as fast as possible. In 50% of our measurements, the entire song was downloaded in 1.6s or less. The maximum download time was 2.01s. Focusing on the playback of playlists, also a fail-safe approach is used. Like in the single song case, the first song is completely downloaded in the beginning. Shortly after the first download finishes, the second song is downloaded while the first one is still playing. The buffer then already contains the next upcoming song, allowing a smooth transition without interruption. Subsequently, the network is idle until the playback of the second song starts. Here the third as well as the fourth songs are fetched. Now the buffer contains the next two songs, instead of only one. At the start of the third song, the fifth is downloaded to sustain the buffer level of two songs. In summary, until the first song has been finished, only the next one has been prepared, by fetching it into the buffer, but after the start of the second song, two future songs are present in the buffer.

The baseline scenario identifies a clear difference in streaming behavior of the Android app compared to the Spotify web player [9]. The app downloads the entire songs as fast as possible and pre-buffers the following two songs to overcome eventual network disturbances. Unfortunately, the service cannot know for sure, which song the user wants to listen to next, even if he is listening to a playlist. He could skip one or two songs or also jump to another position in the list. Fetching too many future songs would generate potentially wasted traffic, which can have a substantial impact if millions of users utilize the service, as it is the case with Spotify. Moreover, each downloaded song has to be stored at the end device, even though it is only temporarily, not all devices may be able to provide sufficient space. Developers have to weight these against one another, before opting for one solution. In contrast, according to [9], the web player only provisions between 10 and 20 seconds of playtime within the buffer. Every time the buffer level hits 10 seconds of remaining song content, the next 10 seconds are requested and downloaded. The web player is mainly used on devices connected to a reliable Wi-Fi or LAN, which includes desktop PCs and laptops. Here the connection is fairly stable, hence planning ahead is not necessary.

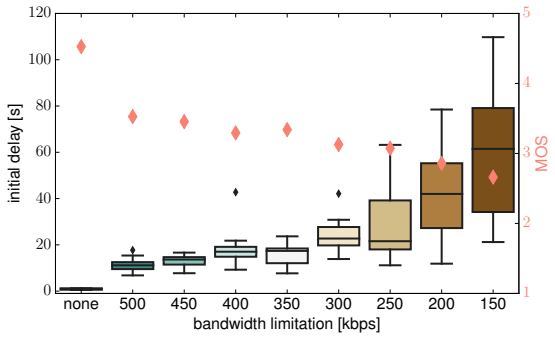

Figure 2: Initial delay

4.2 Bandwidth Limitation Scenario

To analyze the influence of the given network capacity on the streaming behavior, measurements were run under different bandwidth limitations. Here, the focus was set on most important influence factors affecting the QoE of audio streaming, according to [6, 10]: initial delay and stalling time. Furthermore, we evaluated the impact of the bandwidth limitation on the navigation time, as this impacts the user satisfaction as well. In addition to the actual measured delays, the perceived user experience is evaluated by mapping the values to a MOS value according to the model of [6].

Initial Delay. Figure 2 shows the initial delay for different bandwidth limitations. The x-axis describes the limitation of bandwidth in kbps from no limitation up to only 150 kbps for each test series. The y-axis represents the initial delay in seconds, ranging from 0 to 120s. Having no limitation, a comparably low initial delay between 0.51s and 1.20s, with a median of 1.06 seconds is measured. Looking at the limitation of 500 kbps, a relatively huge jump, compared to no restriction, is visible. Here, the median of all iterations of this test series is 11.00s. From a value lower that 200 kbps, the

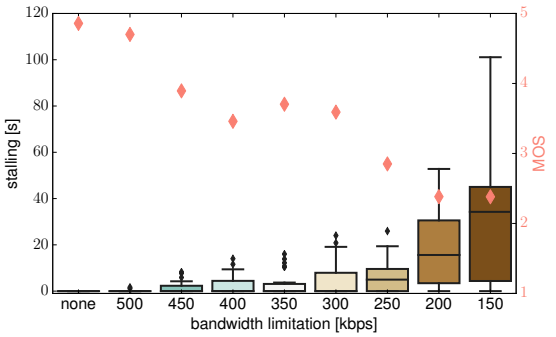

Figure 3: Stalling length

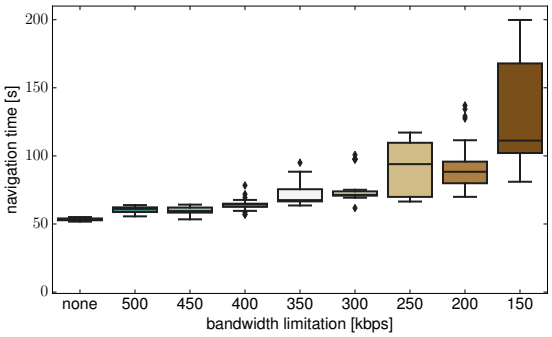

Figure 4: Navigation time

delays drastically increase to a mean of 42.01s for 200 kbps) and 61.47 kbps for 150 kbps. As expected, the initial delays increases for increasing limitations. The Pearson correlation coefficient (PCC) of the medians and the bandwidth limitations emphasize this statement, showing a negative linear correlation of -0.874.

As mentioned before, we used [6] to calculate MOS values from the initial delay and the total stalling time. The result ranges from 5, not annoying, to 1, very annoying and are presented in red in Figure 2. To calculate the MOS value per limitation, each measured value is mapped to its MOS value and afterwards the mean per bandwidth limitation is calculated. Applying their suggested function, no limitation of the bandwidth leads to a MOS value of 4.53, thus a high user satisfaction. Limitations between 500 kbps and 250 kbps result in MOS values between 3.53 and 3.07, which indicates still a fair user experience. For 200 and 150 kbps, the values drop below 3, to 2.86 and 2.66. This concludes that a available bandwidth of 200 kbps or less is unacceptable for the user.

Stalling. The available bandwidth in relation to the total stalling time is shown in Figure 3. Here, again the x-axis displays the bandwidth limitation in kbps and the y-axis the total stalling time in seconds. If multiple stalling occurred in one test, their respective lengths are added together.

For unlimited bandwidth, no stalling occurs at all. The same applies to an available bandwidth of at least 300 kbps. Here, the median of each test series is 0s, thus, in more than half of all iterations, no stalling occur. Limiting the bandwidth to 250 kbps, stalling occur more often and the median stalling time is 4.97s. Increasing the limitation further, also the total stalling time increases. For a bandwidth of up to 200 kbps, the median heavily increases to 15.62s. Having a bandwidth of only up to 150 kbps, the median stalling time is 34.25s. Here again, the PCC of the median of the total stalling time and the bandwidth limitation indicate a negative linear correlation (-0.786). Thus, reduction of the bandwidth results in longer total stalling times.

Having a look at the QoE, the results are more critical than those for the initial delay. When applying the MOS calculation model from [6] on the measured values and calculate the averages per limitation, it becomes clear, that a bandwidth below or equal to 250 kbps is seen as annoying by the users. Having a network capacity of a least 500 kbps, the user satisfaction is close to excellent (MOS of 4.86 for unlimited, 4.70 for 500 kbps). Ranging between 400 kbps and 300 kbps, the users are still satisfied with the service (MOS > 3). For a bandwidth of 250 kbps or less, the total stalling time rises drastically which results in bad QoE values (MOS of 2.85 for 250kbps, 2.39 for 200 kbps, and 2.38 for 150 kbps). Therefore a bandwidth of at least 300 kbps is necessary to keep the users satisfied with the streaming.

Navigation Time. Browsing through the app, which means opening the Spotify app, logging in, adjusting settings, managing playlists, and searching the music database, takes a certain amount of time. Since some interactions require a network connection, they are also directly dependent on the prevailing network quality. An obvious example is using the search bar to find the right song. An unapparent function which requires an Internet connection is, for example, the context menu of an existing playlist. The context menu allows downloading, editing, deleting, sharing, and adding the playlist to the home screen. Additional options are making it secret or collaborative. Intriguingly, when being offline, the context menu also works, without delays, but no cover and no Spotify Code is shown. A Spotify Code is a unique representation of a song, artist, or playlist, which can be scanned by another device using the camera. It seems like the app first attempts to fetch the cover and the Spotify Code, before showing the context menu, which could be the cause for the delay.

Figure 4 shows the measured navigation times for different bandwidth limitations. The y-axis displays the navigation time in seconds. No bandwidth limitation corresponds to a median of 53.12s, which can be seen as an optimum for the navigation time. For a network capacity between 350 kbps and 500 kbps, the median navigation times was 67.38s or lower, and thus, at most only 14.26s (26.84%) slower than the optimal case. Having a look at the navigation time for a limitation of 300 kbps, the value increase drastically, having a median of 71.42s, which is 34.43% slower than the optimal case. This trend continues for increasing limitation, resulting in median navigation times of up to 111.25s for 150 kbps and thus, more than twice as slow as the optimal case.

Unfortunately, to the best of our knowledge, no subjective studies about the navigation times an app are published yet. Nevertheless, it is expected that navigation time of more than one and a half of the required time in the optimal case leads to drastically decrease in the user satisfaction.

5 CONCLUSION

Many studies have already addressed video streaming in great detail. Audio streaming, in contrast, has widely been neglected. Thus, in this work, a measurement methodology has been presented, which allows conducting measurements on Spotify's mobile Android application. The testbed allows extracting relevant application data, like the initial delay, stalling incidents, and the navigation time as well as recording all network activity. To simulate real-world conditions, automatic interactions can be performed and network impairments can be applied. In order to investigate the streaming behavior for suboptimal network conditions, we added different bandwidth limitations to the network.

Considering streaming and buffering characteristics of Spotify's mobile app, we found that for playing one single song, the whole file is directly downloaded into the buffer to overcome network disturbances. Looking at playlists, we found that the mobile app downloads the songs of the playlist in advance to ensure smooth transitions between two songs. Focusing on KPIs for audio streaming, especially initial delay and stalling, the available bandwidth capacity highly influences the user satisfaction. Here, we found that Spotify streaming using the mobile app needs a bandwidth capacity of more than 200 kbps to guarantee at least a fair user satisfaction. Having a look beyond the streaming itself, we also evaluated the user satisfaction by using the app and navigate through it. Therefore, we measured the total navigation time from starting the app until the start of the actual audio playback for different bandwidth limitations and set it in relation to the navigation time with unlimited bandwidth. We found, that for an available bandwidth of 300 kbps or less, the navigation time increase drastically, having a median which is at least 34.43% slower than the optimal case, which presumably leads to a drastic decrease in user satisfaction. Thus, an available bandwidth of at least 300 kbps (3G) is required to make good use of Spotify.

In future work, additional measurements will be conducted using the testbed focusing on other network impairments which can occur in mobile networks, like packet loss or jitter. In addition, as the streaming is not decoupled from the app, it would be interesting to also conduct subjective studies in order to investigate the influence of the navigation time on the perceived user experience.

ACKNOWLEDGMENT

This work is supported by Deutsche Forschungsgemeinschaft (DFG) under the Grant "SDN-enabled Application-aware Network Control Architectures and their Performance Assessment" (project number: 316878574). The authors alone are responsible for the content.

REFERENCES

[1] Goodwater. 2018. Understanding Spotify: Making Music Through Innovation.

[2] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. 2017. A machine learning approach to classifying YouTube QoE based on encrypted network traffic. *Multimedia tools and applications* 76, 21 (2017), 22267–22301.

[3] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. 2011. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM, 25.

[4] Spotify Technology S.A. 2019. Spotify Company Info. https://newsroom.spotify.com/company-info/ Accessed July 4, 2019.

[5] Spotify Technology S.A. 2019. Spotify Reports First Quarter 2019 Earnings, Shareholder Letter. https://s22.q4cdn.com/540910603/files/doc_financials/quarterly/2019/Shareholder-Letter-Q1-2019.pdf

[6] Andreas Sackl, Sebastian Egger, and Raimund Schatz. 2013. Where's the music? comparing the QoE impact of temporal impairments between music and video streaming. In *Quality of Multimedia Experience (QoMEX), 2013 Fifth International Workshop on*. IEEE, 64–69.

[7] Sandvine. 2018. The Global Internet Phenomena Report, October 2018. (2018).

[8] Anika Schwind, Cise Midoglu, Özgü Alay, Carsten Griwodz, and Florian Wamser. [n.d.]. Dissecting the performance of YouTube video streaming in mobile networks. *International Journal of Network Management* ([n. d.]), e2058.

[9] Anika Schwind, Florian Wamser, Thomas Gensler, Phuoc Tran-Gia, Michael Seufert, and Pedro Casas. 2018. Streaming Characteristics of Spotify Sessions. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, 1–6.

[10] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. 2014. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2014), 469–492.

[11] Michael Seufert, Bernd Zeidler, Florian Wamser, Theodoros Karagkioules, Dimitrios Tsilimantos, Frank Loh, Phuoc Tran-Gia, and Stefan Valentin. 2018. A wrapper for automatic measurements with YouTube's native Android app. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 1–8.