How are your Apps Doing? QoE Inference and Analysis in Mobile Devices

Nikolas Wehner*, Michael Seufert*, Joshua Schüler*, Pedro Casas†, Tobias Hoßfeld*

*University of Würzburg, Würzburg, Germany
†AIT Austrian Institute of Technology, Vienna, Austria

Abstract—Web browsing has become the most important application of the Internet for the end user. When it comes to mobile devices, web services are mainly accessed through apps. This paper tackles the problem of Web Quality of Experience (QoE) in mobile devices, with a specific focus on apps QoE monitoring and analysis, using in-network (encrypted) traffic measurements. Measuring apps QoE is complex, not only from an instrumentation point of view, but also from the heterogeneity of user interactions which might realize substantially different user experience. To this end, we conduct a feasibility study on four specific and popular Android apps and their corresponding web services. Our test automation framework emulates and measures different user interactions commonly executed during an app session, including the app startup, clicking, scrolling, and searching. The resulting traffic is characterized on different dimensions, and machine learning models are trained to identify web services, apps, and user interactions, and to infer their QoE. The proposed models can correctly identify the specific web service and app in 86% of the cases and accurately estimate the associated QoE with small errors. Our preliminary study represents a first step towards an in-network, web QoE monitoring solution for mobile-device apps.

Index Terms—QoE; Apps; Machine Learning; Encryption

I. INTRODUCTION

Quality of Experience (QoE) monitoring is an important ingredient for the successful management of mobile networks. It empowers network operators to track the delivery of online services and to rapidly detect issues, which can negatively affect the users' experience. Thus, it allows to ensure a high service quality and a high user satisfaction, while avoiding user churn. In the last decades, web browsing has been one of the users' most popular Internet services. However, web services have evolved in recent years. The majority of users no longer utilizes web services on desktop or laptop PCs, but mostly on smartphones. As reported in [1], smartphone traffic is expected to exceed PC traffic soon, such that 44% of total IP traffic is consumed and generated by smartphones, versus only 19% generated by PCs, in 2022.

This traffic is mainly generated from smartphone applications (apps), which exist in a huge variety and for many purposes, e.g., for news, social media, or web shops. As smartphone applications are based on web technologies, they rely on a composite of multiple multimedia components and embedded services, which makes them different from other services, such as video streaming or gaming. In fact, loading a single web page today requires tens of flows to download the various page resources, which are located in diverse servers from different content providers. In this complex process, the network plays a crucial role influencing users' Web QoE. This urges Internet Service Providers (ISPs) to deploy effective QoE monitoring for web services in general, and for apps in particular when considering mobile traffic, providing wide visibility on meaningful Web QoE-related metrics such as SpeedIndex (SI) [2]. However, metrics like SI require access to the application layer, which is totally hidden from ISPs due to the wide deployment of end-to-end network traffic encryption. Different from our previous work [3], [4] – which focuses exclusively on Web QoE for web browsing, this paper extends previous work by specifically considering smartphone apps for Web QoE monitoring and analysis, which account for a huge share of the traffic in mobile networks.

Measuring (web) apps QoE in smartphones is far from trivial, given all the complexities associated to the instrumentation of QoE measurement in such devices, e.g., lack of APIs for measuring QoE-relevant metrics, a vast heterogeneity of different apps, and others. In addition, capturing the QoE of an app session requires to detect and analyze the different types of user interaction which are taking place.

As a consequence, in this paper we take an explorative, first step into the systematic analysis of apps QoE from an ISP perspective, relying exclusively on the analysis of (encrypted) network traffic. In particular, we report our experience on the feasibility of a measurement and analysis approach to tackle the task, focusing the study on a small set of representative apps. While we cannot claim that our results are general to any other type of mobile app, we believe that the steps taken in the measurement and analysis of apps QoE are fully applicable and reproducible to many others.

For the sake of this study, four popular Android apps and their corresponding web services – i.e., the corresponding websites, accessed through a mobile web browser – are instrumented and measured under various network conditions, using a novel app measurement framework. Different user interactions are emulated and measured, including app startup, clicking, scrolling, and searching, and the resulting traffic is characterized and compared to their respective website counterparts. We show that ML-based website fingerprinting techniques are generally also applicable for apps, resulting in accurate identification of apps/web services and interactions based on simple features. Finally, using predefined Web QoE models based on the SI [5] and ML-based estimation of the SI, we are able to infer the Mean Opinion Score (MOS) for

all app interactions.

The remainder of the paper is organized as follows. Section II overviews the related work on Web QoE monitoring and analysis, focusing particularly on smartphone apps. Section III presents the app measurement framework and the obtained dataset, while Section IV characterizes the selected smartphone applications and their traffic. A fingerprinting approach to identify apps, web services, and user interactions from the encrypted network traffic is outlined and evaluated in Section V. Section VI presents and evaluates the ML-based models for per-app and per-action SI inference, additionally using mapping functions to obtain the associated MOS scores. Finally, Section VII concludes this paper.

II. RELATED WORK

As QoE of web applications is strongly linked to web QoE, loading times [6], [7] are an important factor to infer user satisfaction, e.g, under ITU-T [8]. However, additional metrics have been proposed for web QoE, such as Above the Fold Time (AFT), i.e., the time until the visible portion of a web page has been fully loaded, or SpeedIndex (SI) [2], which considers the whole visual progress of the page loading by processing a video capture of the screen. These metrics have been also tested within traditional Web QoE models [5]. The loading process is influenced not only by network quality fluctuations [9]–[12], but also by usability [13], aesthetics [14], as well as device type, i.e., desktop, smartphone, tablets [15]. Important to our study, these papers show that smartphones and tablets have their own characteristics, not only regarding screen sizes, but also in terms of content rendering and both web and app designs.

Regarding the monitoring of QoE, several tools have been proposed to measure network performance in cellular networks and mobile devices, e.g., Mobiperf [16]. However, most related work requires application layer information, which is problematic for ISPs as the rise of traffic encryption limits the applicability of past approaches relying on DPI [17]. Thus, approximations to metrics like SI have been proposed, such as the Byte/Object-Index [18] and the Pain-Index [19], which can be computed from packet- and flow-level measurements, and thus, seamlessly operate with encrypted traffic. In [20], authors proposed simple modeling approaches to map QoS metrics of app traffic to app-specific QoE metrics.

A prerequisite for accurate Web QoE monitoring and management is to detect and classify the traffic of specific web services and apps. Along some work considering fingerprinting approaches for websites [21]–[23], previous studies have particularly specialized on app fingerprinting. Taylor et al. [24] have proposed AppScanner, which leverages statistical features of packet sizes as input to a Support Vector Machine and a Random Forest classifier. Relying on TCP stream based statistical features, Petagna et al. [25] demonstrate that it is possible to accurately identify apps in traffic anonymized through Tor, and that web browsers play thereby a crucial role as traffic patterns differentiate strongly between browser versions. Finally, the authors of [26] have proposed DECANTeR, which
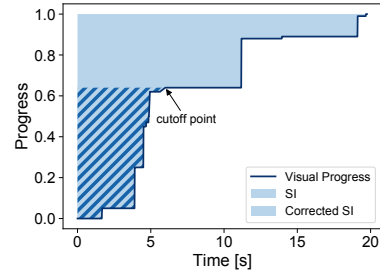

Fig. 1: Correction of the Speed Index.

builds real-time fingerprints from desktop apps exploiting the headers of the HTTP messages.

III. APP MEASUREMENT FRAMEWORK & DATASET

To acquire a data set of network traffic and QoE metrics of several apps in various network conditions, a custom test bed based on the appium[1] framework was built. The multi-device test bed consists of an Android phone and an Android Tablet, controlled by a host computer via the *Android Debug Bridge (adb)*[2], and connected to the Internet over a wireless network, shaped using the *Linux traffic control (tc)*. Appium is a platform independent mobile app automation framework and can be used to remotely control Android apps using the Android-specific *uiautomator*[3]. Appium provides all common user interactions like starting and stopping applications, clicking, and scrolling, as well as several helper functions like finding layout elements or performing a screen capture of the device display. The network traffic is captured using *tcpdump*. As stated before, the SI is the QoE-relevant metric used in the platform, as it can be computed using a screen capture, independently of the underlying user interface technology.

Based on appium, a test automation framework was built in Python, which allows the abstraction of user interactions into distinct consecutive *steps*, each with a separate traffic capture and screen capture. The framework configures the underlying appium instance, provides app setup and teardown functionality, clears data and cache at the beginning of the test and optionally between steps, and configures network traffic capturing and shaping as well as the screen capture. All test parameters are detailed in Table I. The four selected apps include Amazon Shopping, YouTube, Facebook, and BBC News. Each app and each user interaction is tested 50 consecutive times for each of the 12 different network shaping conditions, using both the smartphone and the tablet devices. The app startup from a clean app state is tested for all four apps; additionally, to analyse the impact and behavior of user interactions, several activities like clicking on menu items, searching, and scrolling were performed in the Amazon app.

The resulting data set contains a packet capture in *pcap* format and a screen capture video. To compute the SI from the

---

[1]https://appium.io
[2]https://developer.android.com/studio/command-line/adb
[3]https://developer.android.com/training/testing/ui-automator

PARAMETER	VALUES
Device	Google Pixel 2XL (Android 10)
	Samsung Galaxy Tab S5e (Android 9)
App	Amazon Shopping, YouTube, Facebook, BBC News
Web	amazon.com, youtube.com, facebook.com, bbc.com
Runs	50
Shaping	None
	Bandwidth: 10Mbit/s, 5Mbit/s, 2Mbit/s, 0.5Mbit/s
	Packet loss: 0.1%, 1%, 10%
	One way delay: 5ms, 12ms, 25ms, 50ms

TABLE I: Test parameters.

(a) Number of flows. (b) Traffic volume.

(c) Speed Index on Pixel. (d) Speed Index on Tablet.

Fig. 2: App startup characteristics.

video, first the visual progress for each frame is determined. The first frame is assumed to have 0% visual progress, while the last frame is defined as 100% visual progress. The similarity of frames is based on the alignment of their histograms. Then, the SI is the integral above the visual progress curve as shown in an exemplary way in Figure 1. Two app characteristics observed in our tests inhibited the direct computation of the SI. For one, some views contain animated advertisements, which interfere with the visual progress detection. As these instances were only found in a small portion of test steps, they were manually corrected by ignoring the pixels containing the advertisement when calculating the visual progress. Secondly, a number of views contain slide shows, which also interfere with the visual progress and result in a visual progress curve increasing and decreasing in steps. As these slide shows were a major part of the user interface, ignoring their pixels was not an option. These steps were corrected by only computing the SI to the start of the first plateau of the visual progress function, where the plateau has a threshold length longer than 3 seconds as shown in Figure 1. This ignores the step-like behavior from the periodically changing viewport due to slide shows. In conclusion, each data point in our set is described by the tuple {*run_id, net_shaping, app, action*}, and is labeled by the associated SI value.

To compare the behavior of the apps and their website counterparts, the landing pages of the four websites corresponding to the apps – *amazon.com, youtube.com, facebook.com, bbc.com* – were visited by the same devices over the same network conditions, using the mobile Chrome browser. The network traffic and the SI were captured in the same fashion as for the apps.

IV. APP AND WEBSITE DATA CHARACTERIZATION

This section describes the characteristics of the measured apps on network and application layer for app startups and different app interactions. Additionally, the app measurements are compared to the corresponding website measurements.

A. Comparison of App Startups

App startups are especially important for the analysis, as they represent the initial step in every app session, and are therefore generic to all apps. The characteristics of the app startups are depicted in Figure 2. Considering the network layer, Figure 2a shows the mean number of flows requested by the app at startup when considering all twelve network
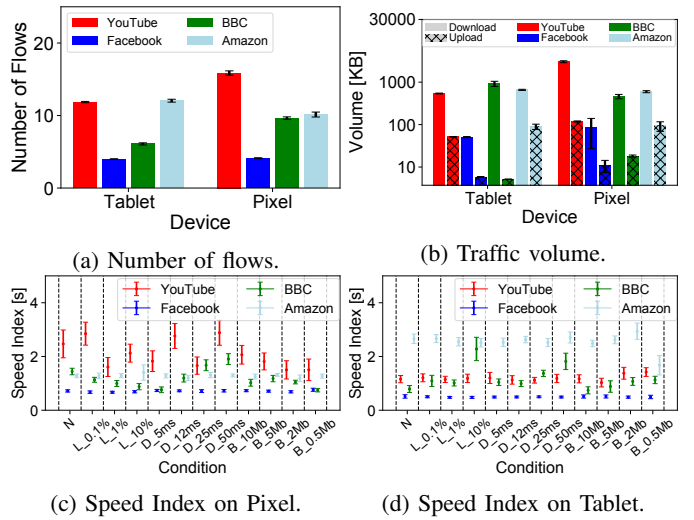
shaping conditions, while the associated total downloaded and total uploaded bytes per app startup are illustrated in Figure 2b. Each bar represents the mean value of the metric, depicted along with its 95% confidence interval. Bars in red represent the YouTube app, bars in dark blue represent the Facebook app, bars in green represent the BBC app, and bars in light blue represent the Amazon app. Upload and download are further distinguished by hatching in Figure 2b, where the hatched bars correspond to the upload volume. Note further that the y-axis depicts the volume in kilobytes on a logarithmic scale.

First of all, it can be observed that there are most of the times visible differences between the app startups on the Pixel and Tablet devices with respect to each metric. Considering the number of flows, the YouTube and Amazon apps require the most flows, while the Facebook app created only four flows on both devices. The behavior of the Facebook app can be attributed to the fact that only the login page, the first impression one could get of the Facebook app when not having an account or not knowing the service, was shown, i.e., no user-specific social media content has been downloaded yet. Interestingly, the BBC app and the YouTube app generated significantly more flows on the Pixel device, while the Amazon app generated a higher number of flows on the Tablet device. This already indicates that there are significant differences in the app behavior on different devices. When contemplating the downloaded bytes for each app startup, there are again strong differences between the devices, in particular for the YouTube app. For the other apps, the differences between the devices are smaller and the download volume is also always below 1 MB. With respect to the upload volume, the Amazon app and the YouTube app account for the most bytes. Again, the difference between the devices is most significant for the YouTube app, while the Amazon app behaves very similar for the startup. The fact that the Amazon and YouTube apps showed the highest data volumes is reasonable as these apps are usually rich of media contents like images or videos,

which are downloaded during startup and where the content is strongly tailored to the end-user, resulting also in higher upload volumes. In contrast, the BBC app usually shows the same content for all users at the startup.

To obtain an additional user-centric view of the app startups, Figures 2c and 2d depict the observed mean SI for each app startup along with the 95% confidence intervals, for both smartphone and tablet. The x-axis denotes the shaping condition and the y-axis denotes the SI. For the shaping conditions, the *N* represents no shaping, the *L* represents packet loss, the *D* corresponds to the one-way packet delay, and the *B* stands for a bandwidth limitation. Further, the appended value and the appended unit of the tick labels specify the degree of shaping. Similar to the network layer, strong differences between apps on the various devices can be observed here too, e.g., the SI for YouTube on the Pixel device is around 2 seconds, while on Tablet the SI varies around 1.5 seconds. Surprisingly, the startup SI values of some apps are basically constant and independent of the network shaping condition. This is mainly true for the app measurements on the Tablet device. As the app cache is deleted manually before app startup, this might indicate that there are some edge caches active, which were out of control for the measurement framework. Besides these exceptions, a more expected behavior can be observed for the remaining cases. In particular, for the bad network conditions with high packet loss (10%) and high one-way delay (50 ms), the SI increases visibly (best visible for the BBC app). Interestingly also is the fact that the strong bandwidth limitations (0.5 MB) do not influence the SI as strongly negative as expected. This indicates that the required bandwidth for an app startup is rather low.

B. Comparison of User Interactions

Next, the characteristics of the interaction steps of the Amazon app are described in detail. Following the characterization of the app startups, Figure 3 depicts the characteristics of the individual interactions. This time, however, the bars and error bars are colored according to the user interaction, whereby orange represents clicking, gray corresponds to the app startup (cf. Figure 2), dark red represents scrolling, and red corresponds to searching.

It is visible that the user interactions click, startup, and scroll show a similar number of flows, while the search interaction results in a significantly lower number of flows. This indicates that a search interaction is a more specific request, addressing only a few servers, while startups, clicking, and scrolling send more generic requests and thus have to fetch contents from multiple servers. Further, the differences in the download volumes of startups, clicking, and scrolling also indicate that app content seems to be not loaded a priori, but that content is only downloaded when requested, which comes as expected. Interestingly though, the total upload volume of an app startup and a scroll interaction are quite similar, while a click interaction results in a significantly higher upload volume. All in all, it can be stated that not the app startup, but single app interactions are responsible for most network

(a) Number of flows.
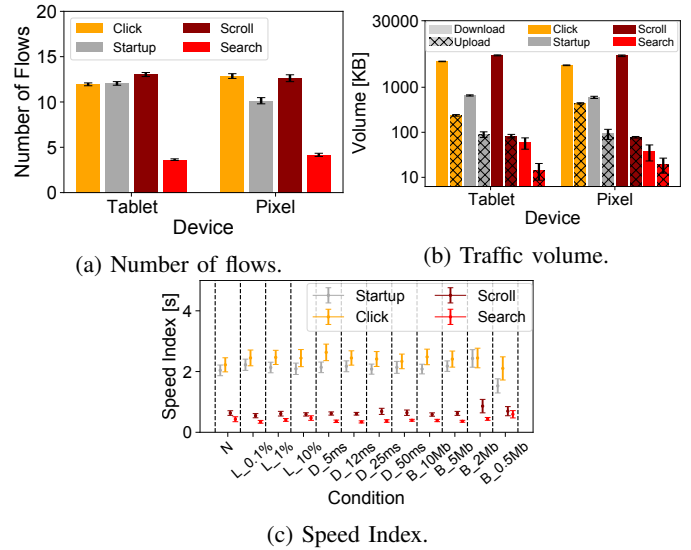
(b) Traffic volume.

(c) Speed Index.

Fig. 3: User interaction characteristics of the Amazon app.

traffic, which emphasizes the necessity of correctly identifying app interactions from the network traffic.

Figure 3c depicts the observed SI for each interaction with the Amazon app and for the different network shaping conditions. Note that this time the runs of both devices are aggregated. Almost all interactions show a stable SI, independently of the applied network shaping condition. In alignment with the amount of generated network traffic, the search interaction with the least network traffic also results in the lowest SI (around 0.5 seconds), directly followed by scrolling. The click interaction and the app startups show a similar, but significantly higher SI of more than 2 seconds, indicating that clicking and app startups may have more impact on the QoE.

C. Comparison to Website Loading

The clean startup of an app can be conceptually compared to the non-cached loading of the associated website. Therefore, the corresponding website loading measurements of an app are characterized in Figure 4. On network layer, Figures 4a and 4b reveal that there are basically no differences between the devices with respect to the number of flows and the observed traffic volume. Further, for all four apps, the traffic volumes are slightly higher for the website requests compared to the app startups, and the website requests result in many more flows than the apps. This shows that website requests and app requests differ strongly for the same application and thus should be handled separately by network providers.

Regarding the SI in Figure 4c, the mean SI computed over all conditions is slightly below 1 second, thus similar to the corresponding app measurements. However, the negative impact of the bad shaping conditions is clearly visible in this case. In particular, when limiting the available bandwidth to 2 Mbps or lower, the SI increases significantly, especially for BBC and YouTube. When analyzing the SI per device,

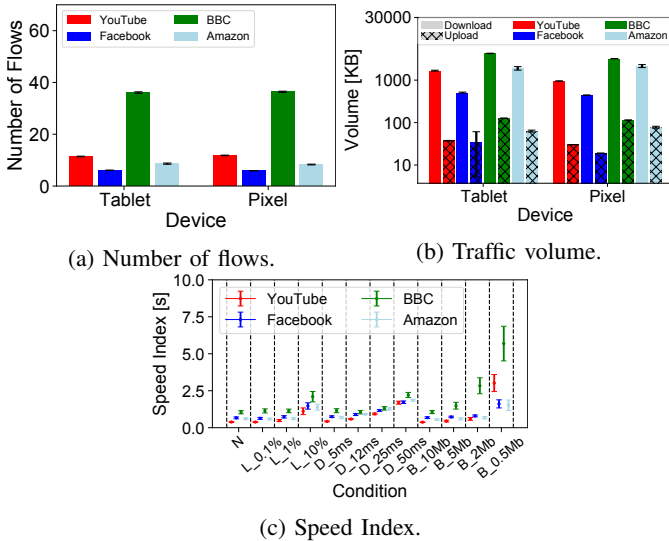(a) Number of flows.

(b) Traffic volume.

(c) Speed Index.

Fig. 4: Website characteristics.

it becomes visible that these low SI values are actually caused by the Tablet device only, and that the SI for the Pixel device barely changed for the majority of shaping conditions. As a consequence and when considering the definition of the SI, this might indicate that the screen size (and potentially other hardware characteristics) is a relevant influence factor for Web QoE when confronted with bad network conditions.

V. FINGERPRINTING

Next, a case study on the identification of apps, user interactions, and web services is performed. Therefore, the upload and download behavior of apps, interactions, and services is investigated. The obtained findings are then deployed for classifying apps, interactions, and services independently, using supervised machine learning techniques.

A. Detection of Apps

In [23], authors propose a method called *Deep Fingerprinting* for solving the problem of website fingerprinting in Tor networks. It is based on the assumption that websites can be distinguished by the sequence of the packets' directions. These sequences are fed as input to a 1D convolutional neural network (CNN), which learns the characteristic patterns of a website request.

Based on this idea, the packet direction sequences generated by the app startups are illustrated in Figure 5a. Note that only the first 50 packets are visualized, as this amount of packets is sufficient to outline differences between apps and websites. Note further that the patterns show the traffic of both devices and all shaping conditions together, which results in some visible variance within the runs for the same app. Each app startup shows a unique traffic pattern, which supports the fact that website fingerprinting methods can also be applied to app fingerprinting. For example, the Facebook app shows the most upload packets relative to the first few packets, while the BBC and Amazon app show the most download packets at the

beginning. In general, the BBC app and the Amazon app show a quite similar behavior for the first 20 to 30 packets, before the BBC app starts generating a more regular consecutive upload and download pattern compared to the Amazon app.

As proposed in [23], a 1D CNN could now be used to identify the apps. However, as the dataset comprises only four apps, a neural network would severely overfit. To avoid this and for the sake of simplicity, a simple linear classifier is taken for the identification of the app startups. As app classification formulates a multi-class classification problem, softmax regression is used here. The data is split into 80% training data and 20% test data beforehand and 5-fold cross validation is used for training. Using the number of download packets for the whole run and for the first 50 packets, the number of upload packets for the whole run and for the first 50 packets, and the number of flows as features, an accuracy of 93% is achieved. The corresponding confusion matrix is depicted in Figure 6a, where each number indicates the number of performed predictions. Most misclassifications occurred for the Amazon and BBC apps, resulting in a F1 score of 85% for the Amazon app and a F1 score of 88% for the BBC app. In contrast, almost all instances of the Facebook app and the YouTube app are classified correctly. These results show that even with a simple model and five features only, a reasonable classification performance can be obtained. However, for large scale app monitoring this approach will probably not suffice.

B. Detection of User Interactions

The same scheme is now applied for the user interactions. Figure 5b presents the observed sequences of packet directions for the individual interactions. Remember that only the Amazon app is investigated here, so the startup is identical to the Amazon pattern in Figure 5a. Again, there are obvious differences in the patterns for each interaction. It can be seen that scrolling results in the most regular pattern, while the click interaction results in a much more irregular pattern.

Again, softmax regression and the same features as before are leveraged for the interaction identification, resulting in an accuracy of 93%. The resulting confusion matrix is shown in Figure 6b. The identification of the startup interaction and the search interaction hereby result in a F1 score of 96% and 94%, respectively, while most misclassifications happen for the click interaction (F1 score: 90%). These instances are mistaken for scrolling or searching, which can be explained by the highly variant traffic pattern of the click interactions observed before.

C. Detection of Services and corresponding Apps

Last but not least, the requested service and the corresponding app, i.e., website request or app request, are discriminated from the encrypted network traffic. The traffic patterns (first 50 packets) for the website measurements are depicted in Figure 5c. The figure depicts a similar traffic pattern for all websites, resulting in less visible differences between the websites. This similarity can be attributed to the fact that the TCP handshakes and the TLS handshakes for each website
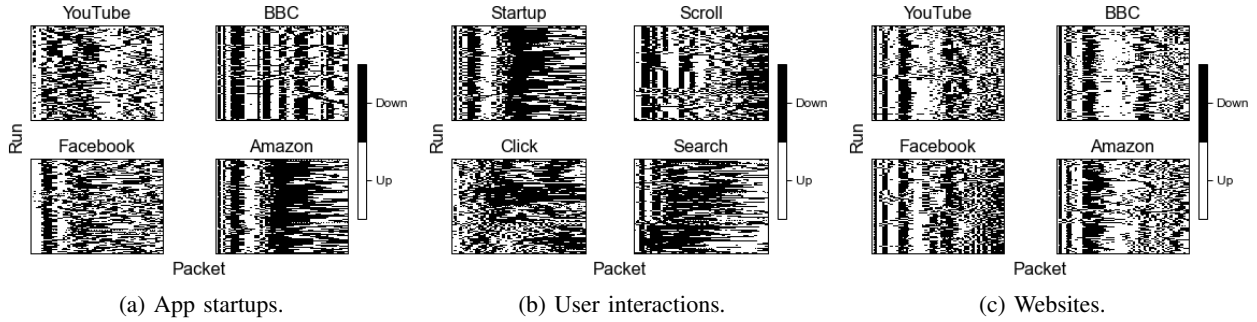
(a) App startups.

(b) User interactions.

(c) Websites.

Fig. 5: Traffic patterns showing the observed sequences of upload and download packets.

(a) Identification of apps.

(b) Identification of interactions.
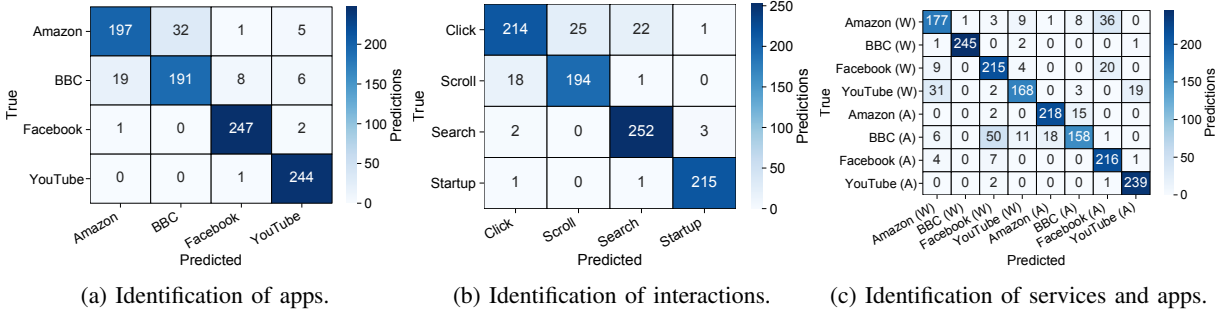
(c) Identification of services and apps.

Fig. 6: Fingerprinting of apps, interactions, and services.

request are depicted here. When investigating more than the first 50 packets, again differences between the websites can be observed. This reveals that the services app and web already differ in the way they setup their network connections.

To discriminate apps in dependency of the used service, the same methodology is used as before. The resulting confusion matrix is shown in Figure 6c, where a *W* represents the web service, and an *A* represents the app service. The evaluation resulted in an overall accuracy of 86%, where most misclassifications happen for the BBC app (recall of 69%). In contrast, the BBC website is correctly identified in almost all cases. All in all, all services and apps can be identified with a precision of at least 79% and a recall of at least 75% (except for the BBC app).

VI. SPEED INDEX INFERENCE

The last goal of this work is the accurate inference of the QoE of app startups, app interactions, and web requests from encrypted network traffic. The findings regarding the relationship between the SI and the Mean Opinion Score (MOS) from [5] are hereby leveraged to map the SI to the assumed QoE. The authors observed a logarithmic relationship between the Byte Index (a proxy to the SI) and the 5-point MOS scale ranging from 1 (bad) to 5 (excellent). Their best model fitting $f(t) = -0.4731 \cdot ln(t) + 7.0813$ for this relationship is thus here used to compute the QoE from the SI. The obtained MOS values are then fed as targets into a well-known Decision Tree model. The number of flows, the total download bytes, and the total upload bytes serve as input features and the model is built to predict the QoE
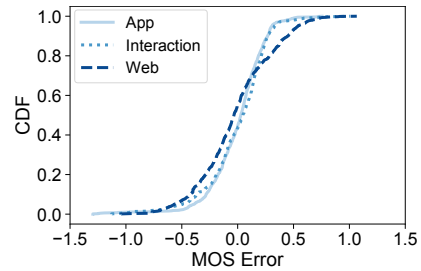

Fig. 7: QoE inference results.

(MOS score) as a regression task. Three different models with different use cases are trained. In particular, one model for the app startups, one model for the interaction steps, and one model for the websites are trained separately. All data is split into 80% training data and 20% test data and to find the best configuration, 5-fold cross validation with respect to the mean squared error is performed. The Decision Tree is further regularized by setting the maximum depth of the tree to three.

Figure 7 depicts the results for the different models and use cases in form of a CDF. The x-axis depicts the MOS inference error and the y-axis depicts the corresponding fraction of runs with an error equal or lower to x. It can be stated that the inference error for the app startup QoE (solid light blue line) is most of the time between -0.5 and 0.5, an acceptable boundary with respect to the MOS scale. Further, the root mean squared error (RMSE) is around 0.24 and the median absolute error (mAE) is around 0.14, indicating good results in general. Considering the QoE inference for the Amazon

app interactions (dotted line), it can be observed that the obtained error is here slightly larger than for the app startup model, resulting in a RMSE of 0.28 and a mAE of 0.16. When investigating the interactions independently from each other, these negative deviations can be traced back to the click interaction and the scroll interaction in particular. In contrast, the impact of the Amazon startup on the QoE inference seems negligible. This finding suggests that the app interactions besides the app startup complicate a reasonable QoE inference severely. Finally, similar results as for the interactions are obtained for the web measurements (dashed dark blue line) with the RMSE being around 0.33 and the mAE being around 0.21. These findings indicate that the inference of app startups is easier than for websites. One possible reason for this might be the overhead for setting up the network connections (as shown in Section V), potentially resulting in more noisy traffic. However, this needs to be checked in a large scale study in future work.

VII. Conclusion

This work measured four popular Android apps using a novel app measurement framework and their corresponding web pages under various network conditions. During the measurement, different user interactions were emulated within the apps, such as app startup, clicking, scrolling, and searching. The measurements showed a strong difference between apps on different devices, both due to different network layer behavior and due to the influence of the screen size on the SI. Similarly, we observed a strong difference in traffic patterns between using a web service from an app or from a browser, highlighting the importance of apps for a holistic QoE assessment in mobile networks.

To identify services and infer Web QoE from the encrypted stream of packets, we trained machine learning (ML) based models on the measurement data. The discriminate traffic patterns allowed to distinguish the apps and web services with a very high accuracy and to estimate the app QoE with small errors using simple features. In future work, these results have to be confirmed in a large scale study using more measurements of a larger set of applications from more mobile devices and in more diverse network conditions.

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017–2022," 2019.

[2] Google, "Speed index." [Online]. Available: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

[3] N. Wehner, M. Seufert, J. Schuler, S. Wassermann, P. Casas, and T. Hossfeld, "Improving web qoe monitoring for encrypted network traffic through time series modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 4, p. 3740, May 2021.

[4] S. Wassermann, P. Casas, Z. B. Houidi, A. Huet, M. Seufert, N. Wehner, J. Schler, S. Cai, H. Shi, J. Xu, T. Hossfeld, and D. Rossi, "Are you on mobile or desktop? on the impact of end-user device on web qoe inference from encrypted traffic," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–9.

[5] T. Hoßfeld, F. Metzger, and D. Rossi, "Speed Index: Relating the Industrial Standard for User Perceived Web Performance to Web QoE," in *Proceedings of the 10th International Conference on Quality of Multimedia Experience (QoMEX)*, 2018.

[6] E. Ibarrola, I. Taboada, R. Ortega *et al.*, "Web qoe evaluation in multi-agent networks: Validation of itu-t g. 1030," in *2009 Fifth International Conference on Autonomic and Autonomous Systems*. IEEE, pp. 289–294.

[7] S. Egger, T. Hoßfeld, R. Schatz, and M. Fiedler, "Waiting Times in Quality of Experience for Web Based Services," in *Proceedings of the 4th International Workshop on Quality of Multimedia Experience (QoMEX)*, Yarra Valley, Australia, 2012.

[8] International Telecommunication Union, "ITU-T Recommendation G.1030 : Estimating End-to-end Performance in IP Networks for Data Applicationss," 2014.

[9] A. Sackl, P. Casas, R. Schatz, L. Janowski, and R. Irmer, "Quantifying the impact of network bandwidth fluctuations and outages on web qoe," in *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, pp. 1–6.

[10] A. Saverimoutou, B. Mathieu, and S. Vaton, "A 6-month analysis of factors impacting web browsing quality for QoE prediction," *Computer Networks*, vol. 164, p. 106905, 2019.

[11] A. S. Asrese, S. J. Eravuchira, V. Bajpai, P. Sarolahti, and J. Ott, "Measuring web latency and rendering performance: Method, tools, and longitudinal dataset," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 535–549, 2019.

[12] M. Rajiullah, A. Lutu, A. S. Khatouni, M.-R. Fida, M. Mellia, A. Brunstrom, O. Alay, S. Alfredsson, and V. Mancuso, "Web experience in mobile networks: Lessons from two million page visits," in *The World Wide Web Conference*, 2019, pp. 1532–1543.

[13] M. Varela, L. Skorin-Kapov, T. Mäki, and T. Hoßfeld, "Qoe in the web: A dance of design and performance," in *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, pp. 1–7.

[14] M. Varela, T. Mäki, L. Skorin-Kapov, and T. Hoßfeld, "Towards an understanding of visual appeal in website design," in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, pp. 70–75.

[15] S. Baraković and L. Skorin-Kapov, "Survey of research on quality of experience modelling for web browsing," vol. 2, no. 1, p. 6.

[16] "Mobiperf, Measuring Network Performance on Mobile Platforms," Accessed: January 11, 2016. [Online]. Available: https://sites.google.com/site/mobiperfdev/home

[17] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, pp. 295–312.

[18] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," vol. 46, no. 4.

[19] M. Trevisan, I. Drago, and M. Mellia, "PAIN: A Passive Web performance indicator for ISPs," vol. 149.

[20] A. Nikravesh, Q. A. Chen, S. Haseley, X. Zhu, G. Challen, and Z. M. Mao, "Qoe inference and improvement without end-host control," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 43–57.

[21] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[22] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[23] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 19281943.

[24] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 439–454.

[25] E. Petagna, G. Laurenza, C. Ciccotelli, and L. Querzoni, "Peel the onion: Recognition of android apps behind the tor network," in *International Conference on Information Security Practice and Experience*. Springer, 2019, pp. 95–112.

[26] R. Bortolameotti, T. van Ede, M. Caselli, M. H. Everts, P. Hartel, R. Hofstede, W. Jonker, and A. Peter, "Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting," in *Proceedings of the 33rd Annual computer security applications Conference*, 2017, pp. 373–386.