Tackling Mobile Traffic Critical Path Analysis With Passive and Active Measurements

Gioacchino Tangari
University College London
gioacchino.tangari.14@ucl.ac.uk

Diego Perino
Telefonica Research
diego.perino@telefonica.com

Alessandro Finamore
O2 - Telefonica UK
alessandro.finamore1@telefonica.com

Marinos Charalambides
George Pavlou
University College London
firstname.lastname@ucl.ac.uk

***Abstract*—Critical Path Analysis (CPA) studies the delivery of webpages to identify page resources, their interrelations, as well as their impact on the page loading latency. Despite CPA being a generic methodology, its mechanisms have been applied only to browsers and web traffic, but those do not directly apply to study generic mobile apps. Likewise, web browsing represents only a small fraction of the overall mobile traffic. In this paper, we take a first step towards filling this gap by exploring how CPA can be performed for generic mobile applications. We propose Mobile Critical Path Analysis (MCPA), a methodology based on passive and active network measurements that is applicable to a broad set of apps to expose a fine-grained view of their traffic dynamics. We validate MCPA on popular apps across different categories and usage scenarios. We show that MCPA can identify user interactions with mobile apps only based on traffic monitoring, and the relevant network activities that are bottlenecks. Overall, we observe that apps spend 60% of time and 84% of bytes on critical traffic on average, corresponding to +22% time and +13% bytes than what observed for browsing.**

I. Introduction

Web browsing has been at the core of Internet services since its early days. Significant attention has been devoted to define metrics [6], [7], [21], [38] and methodologies [12], [16], [21], [39] to unveil webpages content delivery dynamics, and systems to optimize content delivery [8], [22], [24], [40]. These efforts are justified to improve end-users quality of experience (QoE), while service providers are incentivized to optimize their systems as their revenues are linked to users QoE.[1] Within these works, Critical Path Analysis (CPA) studies the delivery of webpages to identify page resources, their interrelations, and their impact on the user experience. To this goal, CPA first identifies a delivery deadline capturing the user QoE. For instance, the most widely adopted metric is *Page Load Time* (PLT), the time elapsed between a user clicking a URL and the browser triggering the `onLoad` event. Once the deadline is defined, CPA investigates how each object download, parsing, and rendering tasks can be a bottleneck for the overall webpage delivery [8], [39], [40].

The role of browsing is however shifting, as it is not at the center of user activities on mobile devices anymore. Recent reports [5], [11] show that users spend less than 10% of their time browsing, and more than 35% on apps different than Facebook, streaming, gaming, and instant messaging. Such a trend is challenging also ad platforms where browsing on mobile devices generates half the conversion rate of desktops [10], [27]. This progressive change in user interests and usage patterns is creating a *gap in the literature*, and CPA has not been investigated in the context of generic mobile apps. At high level, CPA is based on three requirements: i) define a delivery deadline capturing the user QoE; ii) characterize network activities and their relationships iii) identify which activities impact the delivery deadline, thus users QoE.

To address the first requirement we need to answer the questions: "*Is any of the performance metrics defined for web traffic suited as delivery deadline for generic mobile apps? If not, how can we define a more suitable metric?*". The literature has proposed several delivery deadlines. Despite PLT is the commonly used metric, *Above The Fold* (AFT) [7] and *Speed Index* (SI) [17] are considered superior. Both have been introduced by Google and focus on rendering dynamics measured via a screen capture. Their costs is not negligible, and confine their adoption only to properly instrumented devices. Trying to solve AFT/SI constraints, the research community has proposed a flourished set of alternatives (Yslow, Object Index, DOMLoad, etc. [6]). As for PLT, these metrics are easier to compute than AFT/SI, but they are all web traffic specific. Conversely, AFT and SI do not make assumptions on content or application internals, so they are applicable to study generic mobile apps.

To understand traffic dynamics, identifying the right metric is not enough. Indeed, within the boundaries of each delivery deadline, a second question to address is "*How to identify which flows carry critical content for the overall delivery, without knowing the properties of the content itself?*". For instance, when considering web traffic, CPA leverages a *dependency graph* where nodes represent the content downloaded while edges map the interdependecies between objects and related activities (e.g., parsing, script execution, rendering). While extracting such graph can be trivial for webpages (e.g., parsing webpages source code, or inspecting the document object model - DOM), there is no guarantee that mobile content is served in the form of a webpage. It is therefore not clear how different network activities can be identified and how to understand which ones impact the delivery deadline.

Finally, another critical question is: "*Where is CPA performed?*". The common choice in the literature [9], [40], [32] is an in full control scenario (e.g., rooted device, apps source

[1] https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales

code, operating system "modding"). In this case we can obtain fine-grained info at the cost of limiting the study to a few apps and users. Conversely, in the interest of enabling at-scale studies, we aim to study how to reduce the amount of on-device instrumentation in favour of a more traffic measurements centric approach.

In this paper we present our journey in answering the previous questions. We introduce a methodology, namely *Mobile Critical Path Analysis* (MCPA), which combines passive and active measurements and heuristics to: i) recognize user interactions with applications; ii) extract network activities and relevant delivery deadlines; iii) identify network traffic that is critical for performance. To understand the effectiveness of MCPA we follow the standard practice of using an instrumented device, and we dissect the traffic of 18 popular mobile apps. Results show that via passive measurements we can identify user engagement with more than 80% accuracy (§V). We can also define delivery deadlines at least as good as PLT for browsing, and with less than 1.3s of error with respect to AFT in 75% of tests with other apps (§VI). By means of active experiments, we can split traffic into "phases" likely related to apps logic. Overall, we find that apps spend 60% of time and 84% of bytes on critical traffic on average (§VII). Results support the idea of a new class of tools based on network measurements that ease the study of mobile apps. MCPA source code and experimental datasets are publicly available.[2]

II. Related Work and MCPA challenges

Mobile traffic has mostly been studied at an aggregated level (per-connection latency, throughput, etc.) [14], [23], [28], [33], or focusing on specific protocols (e.g., DNS [4], SPDY [13], MPTCP [19], [20]). Exceptionally, a few studies take a step further. For instance, Panappticon [41] and AppInsight [32] enable fine-grained view on users engagement with apps by respectively tapping into Android components and studying app binary files; QoE Doctor [9] focuses on performance issues (e.g., high latency) by measuring radio resource allocation and user interface interactions; Prometheus [2] tries to bridge network metrics with user experience via machine learning.

Despite their merits, these tools focus on system information (e.g., radio resources, operating system calls, multi-threading) rather than digging into the role of content download and network protocols dynamics. Conversely, studies focusing on web traffic, despite being limited to this traffic class only, represent the state of the art regarding how to dissect traffic dynamics. In the remainder of this section we review this literature, and we highlight the challenges in applying currently available methodologies to study generic mobile traffic.

A. Performance metrics and delivery deadlines

Beside generic metrics such as latency and throughput, most of the metrics in literature are defined in the context of web traffic. We can split those into two classes: *objective metrics* are delivery deadlines quantifying the time needed to obtain some content [1], [13], [14], [25], [30]; *subjective metrics* are defined considering direct feedback from end-users (e.g., mean opinion score - MOS) and can include factors beyond content delivery [6], [15], [22], [37]. For the purpose of this work, we focus only on objective metrics, which we can further split into *time instant* and *time integral* metrics.

Time instant metrics capture specific instants across the whole events timeline of the content delivery. The most accurate instant metric is Google's AFT which measures the time at which the content shown in the visible part of a webpage is completely rendered [7]. This definition is not web traffic specific, although the metric has been applied only to browsing traffic. AFT computation requires a video screen capture, and accurate video post-processing as the presence of dynamic elements, such as animations and roll ads, can introduce biases [37]. These costs limit the use of AFT for small scale studies on instrumented devices. A recent work shows that AFT could be approximated leveraging information about objects position in a webpage, but this technique is complex to be applied outside browsers [12]. Despite being less accurate, PLT is the most widely adopted metric. Other known deadlines are the *Time To First Byte* (TTFB), the *Time To First Pixel* (TTFP), the time at which the parsing of the *Document Object Model* (DOM) is completed. W3C has also defined the navigation timing guidelines [38], a series of specific events happening during a webpage rendering, but their implementation may differ across browsers.

Time integral metrics capture the cumulative effect of events until a specific point in the timeline is reached. The most popular example is Google's SI [17], which is obtained by integrating over time the residual rendering left to reach the AFT. Given the definition, SI suffers from the same limitations AFT does. ObjectIndex and ByteIndex are two alternative integral metrics that respectively capture the evolution of objects and bytes delivery until the PLT [6].

Challenges: Metrics like PLT, which are based on internal application "hooks", cannot be applied to generic mobile apps as there are no standard APIs, neither at app nor at operating system level, to expose these information. Differently, we argue that *AFT and SI are valid delivery deadline for generic mobile apps*, as they capture the actual screen rendering and do not depend on app internals (§III). However, their measurement cost is a barrier for their adoption. To enable at-scale measurement, a cheaper alternative is to opt for metrics based on *passive traffic measurement* to compute either on-device (e.g., via VPN solution which avoid rooting devices) or in-network (e.g., monitoring middle-boxes are very common in mobile networks). We are therefore interested in understanding *what passive metrics are available, when they can be applied, and what bias they introduce with respect to AFT and SI.*

B. Critical Path Analysis - CPA

CPA allows to dissect traffic dynamics within the boundaries of a delivery deadline. It has been successfully applied to understand web traffic, but methodologies and terminology can vary. To the best of our knowledge, the first tool leveraging

[2] https://www.dropbox.com/sh/rk853z5e4911mjy/AADOvq19EQ05R5ZDK_Jt0w1Ya?dl=0

CPA is WProf [39] (and its follow ups Shandian [40], and WProfX [29]), a system that requires augmenting the browser with a profiling engine to capture the *dependency graph* for any given webpage. Such graph structures the activities related to both rendering as well as content dependencies as visible in the webpage DOM. Given a graph, WProf defines the critical path as the longest path of activities such that reducing the duration of any activity not on the critical path does not impact the webpage PLT.

Recently, Google added Lighthouse [18] to the Chrome devtools suite to automate webpages auditing. Lighthouse offers a richer output than WProf, including different deadlines (First Meaningful Painting, First CPU idle, SpeedIndex, etc.), as well as a report on resources that can block the rendering. To some extent, Lighthouse output is an evolution of a webpage *download waterfall*, *i.e.,* a gantt chart picturing the evolution of the network communications triggered during a webpage load. All modern browsers allow to dissect traffic dynamics via a waterfall, and systems like KLOTSKI [8] further build on waterfalls to find activity patterns invariant to PLT performance.

Challenge: All these tools have slightly different critical path definitions. They also heavily rely on "hooks" specific to browsers internals, so they are unappealing to study mobile apps. At the core of CPA there is the need to identify dependencies between activities, and this is particularly challenging to do only based on passive measurements. Hence, we want to understand if *active experiments*, such as traffic throttling, can complement passive measurements to create a more effective methodology to spot traffic impacting the delivery deadline.

III. MCPA Overview

In this section, we introduce MCPA, our methodology to perform CPA on generic mobile apps. First, MCPA identifies activity windows, *i.e.,* user interactions with apps. Each activity window is profiled to extract network activities, measure the delivery deadline, and finally extract the critical traffic.

Activity windows (§V). In the context of web traffic, CPA is performed for every webpage retrieval. This includes all activities in response to directly typing a URL, refreshing or aborting the load of a webpage, clicking a link within a page, etc. For webpages, those activities can be easily identified using APIs provided by browsers. However, such mechanisms are not available to study generic mobile apps, so alternative approaches need to be considered. One option is to log user clicks, scrolls, currently displayed apps, and use such detailed information to partition the traffic based on user engagement. However, in an at-scale scenario, *i.e.,* without full control on the devices, logging actual user interactions is almost impossible. Another option available is to apply "cheaper" passive traffic analysis heuristics. In fact, mobile traffic is bursty in nature [14], [35], *i.e.,* the traffic presents *activity windows* when the user is interacting with the phone, interleaved by "idle" periods. An optimal split associates a different user action to each window, but depending on traffic conditions and apps characteristics this might not always be possible. In §V we discuss heuristics for partitioning the traffic based on passive measurements and we evaluate their accuracy.

Download waterfall and performance metrics (§VI). For each activity window we need to define a set of metrics and identify the activities involved in the delivery of contents. CPA for webpages requires to instrument the browser to extract all activities participating to both the download and rendering tasks. However, to do the same for generic mobile apps would require to either reverse engineer every app, or to instrument their source code or the operative system [32], [41]. The approach of MCPA is to focus only on network activities and to report per-flow metrics for both transport (TCP, UDP, QUIC) and application (DNS, HTTP, HTTPS/TLS, Facebook Zero - FB0) protocols. These activities are visually represented in the form of a download waterfall.

Once the different activities are identified, a delivery deadline should be set to capture the quality of experience perceived by users. In a fully controlled environment, the best available option is to apply AFT and SI (§II). We argue they are still valid to study generic mobile traffic, but we are not aware of any work in the literature proving this. Indeed, the end of a user action on an app is generally marked by visual changes, and this applies to apps wrapping browser(-like) functionalities (e.g., social, news, e-commerce), as well as to more interactive apps such as messaging ones (e.g., the end of a message delivery triggers a check mark on screen). However, both AFT and SI capture events related to rendering. In an at-scale scenario screen recording is not possible, so rather than looking for exact estimates of user experience, we are interested in defining a proxy for AFT/SI, yet sufficient to identify critical activities, based on passive measurements. In §VI we discuss how MCPA creates waterfalls, we introduce our delivery deadlines, and we compare them against AFT/SI.

Critical Path (§VII). Finally, MCPA identifies which activities of a waterfall constitute the critical path. To do so, we rely on active experiments, *i.e.,* we observe how the delivery deadline changes when throttling the traffic on a per-domain basis. In other words, if a macroscopic delay is observed on the overall delivery when delaying some traffic, we can conclude that a domain, and the related traffic, is critical. The same principle also applies to discover relationships among domains.

MCPA is built upon `pcap2har`, a Python open source tool transforming pcap files into webpages HAR files,[3] which we modified and extended to handle generic mobile traffic (including TLS/HTTPS, QUIC, FB0).

IV. Dataset

Mobile Apps. We select 18 popular apps across 7 categories: Social (Twitter, Facebook, Instagram), Messaging (WhatsApp, SnapChat, Messenger), News (CNN, BBC, Newsbreak), Geo-based (Google Maps, Uber), Shopping (Letgo, Amazon), E-mail (Microsoft Outlook, Gmail), and Streaming (Youtube, Spotify, Soundcloud). We intentionally left out Games and Productivity apps as they are known to generate little network

[3] https://github.com/andrewf/pcap2har

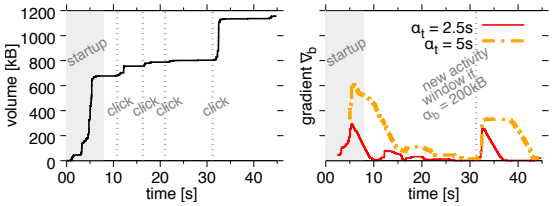

Fig. 1: Activity windows: cumulative traffic when using the CNN app (left); traffic gradient ∇_b (right).

traffic, which is likely related to ads [3]. Conversely, we focus on very popular apps according to both vendors [34], and 3rd party[4] rankings, to create a set of apps sufficiently diversified to assess if there is a case to use passive and active analysis to perform CPA. We further consider web browsing by studying the top-100 Alexa websites (alexa-T100).

Traffic Scenarios. We consider two traffic scenarios: app-startup and app-click. The former considers the traffic generated in the first 60s after the app is launched.[5] In the latter, relevant user interaction sequences are emulated based on common behaviors with the apps, such as select a video/song, a news, scroll an email, send a chat message, etc. To this end, we define ad-hoc patterns, each with multiple `input tap` events uniformly distributed within [0,10s]. For example, for the Letgo shopping app, the sequence is: search by category; show top results; select random item; show price and geographical location (all sequences listed in [36]).

Data collections. Our experiments are performed on a Nexus 5 running Android 6.0.1, and using a SIM of a European mobile carrier. For each app and scenario we ran 10 experiments, with the device instrumented to collect pcap files (via `tcpdump`) as well as the video screen record (via Android `screenrecord` utility[6]). For alexa-T100 dataset, we also use WProfX, Google Lighthouse and Chrome's devtools to extract performance indicators and critical path information. In regards to video recording, as shown in [6] the additional computation can bias the experiments, artificially slowing the rendering. We verified that this effect is not present in our results (details in [36]).

V. Activity Windows

Mobile devices are constantly connected to the network, so they generate a continuous stream of connections. Conversely, user engagement is occasional, hence the connections stream has to be processed in order to identify those time intervals where users interact with the device. Ideally, the traffic stream should be split so that each partition corresponds to a relevant QoE-related user interaction. We call these partitions *activity windows*. Such windows can be obtained using granular device-screen logs reporting on clicks, scrolls, etc., at the cost of running tests only on a limited set of instrumented phones.

To enable large scale analysis built on network measurements, the same split should be performed by looking at traffic characteristics only. To this end, we can exploit the bursty nature of mobile traffic, where bursts of bytes are likely to correspond to user engagement with an app. For instance, consider Fig 1(left) showing the cumulative traffic observed when a user interacts with the CNN app. Notice how volume abruptly increases in response to users actions. In this section we investigate how and to what extent traffic bursts and idle periods can be used to identify activity windows.

A. Partitioning policies.

We consider two possible policies to partition the traffic generated by a mobile device.

Naïve. The first policy relies on a single threshold to identify "long" idle periods. That is, a connection is associated to a new window if its traffic starts after an idle period longer than α_t, otherwise it belongs to the current window.

Gradient. A more refined policy creates a new window if a "large" burst happens after a "long" idle period. To do so, we combine two thresholds: α_t and α_b. We use α_t to define a sliding window where we monitor the *gradient* ∇_b of the volume. For instance, consider α_t = 5s. All traffic in the first 5s is accumulated. Then, we progress the sliding window, accumulating the traffic entering, and removing the one falling outside the window. In this way ∇_b has a positive slope when traffic is exchanged, and negative (or no) slope for idle times. Fig. 1(right) reports ∇_b for α_t = 2.5s and α_t = 5s. Using the gradient, we define a new activity window if we observe at least α_b bytes exchanged after an idle period of α_t. For instance, considering α_b = 200kB, in Fig. 1(right) ∇_b reaches the threshold at 5.2s and 32s. However, we identify an activity windows only at 32s as it is preceded by an idle larger than α_t = 2.5s (no windows found for α_t = 5s).

B. Validation and sensitivity analysis

Our dataset contains detailed logs of the users click times, each click corresponding to the beginning of a new activity window. As such, for a given combination of thresholds, we can quantify the accuracy of the partitioning by measuring the *Precision* as the fraction of partitions detected by our policies actually matching a click, and the *Recall* as the fraction of clicks that are identified as activity windows by our policies. For instance, in Fig. 1 Precision = 1.0 and Recall = 0.25.

Best policy. We find the naïve policy being ineffective. In fact, a small threshold (α_t <1s) leads to over-splitting (high Recall, but low Precision), while for larger values Recall and Precision do not go above 50% (see [36] for details). Compared to naïve, the gradient policy, which considers bursts registered after idle periods, significantly reduces the over-splitting. By selecting α_b = 5kB and α_t = 1s, both Recall and Precision are kept above 70%. We choose α_b to be the median size of a single transaction as observed in logs from a large European mobile operator (see [36]), while α_t = 1s is considered as a minimum response time of a user engaging with mobile apps.

Further improvements. Performing a grid search to find thresholds better than the ones selected based on our intuition did not help. However, we found most of the misclassification

[4]https://www.androidrank.org

[5]This time is more than double the maximum startup time observed in our experiments.

[6]https://developer.android.com/studio/command-line/adb

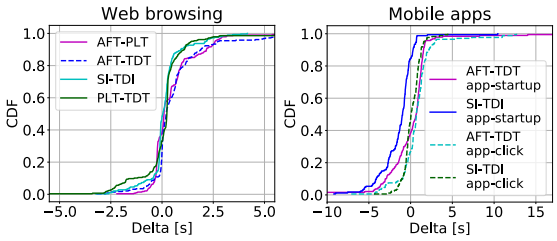

Fig. 2: TDT and TDI accuracy evaluation

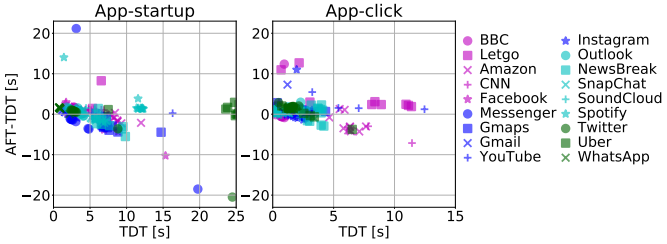

Fig. 3: Per-app instant metrics comparison

are due to chat apps. Intuitively, as those apps typically exchange small messages (unless they are video/audio messages, or images), α_b = 5kB is too large. Indeed, applying α_b = 0.25kB only for this app category leads to Recall = 85% and Precision = 88% across all apps. Although these fine-grained optimizations could be done on a per-app basis, we argue this is unnecessary, and would be also challenging considering the large numbers of apps currently available. In fact, even if our analysis is not exhaustive, two pairs of thresholds cover a very diversified set of apps. In order to select which pairs of thresholds to use, we found that basic traffic classification techniques, based on port numbers, IP addresses, or domain names, are sufficient. For instance, chat applications use very few (and specific) domains and/or ports (§VII).

Background traffic. One last aspect to consider is the impact of "background" traffic (notifications, emails fetch, etc.) on the windows partitioning accuracy. We collected several 1-day long traces, mixing periods of activity with silence. We observe that, while the gradient policy is still sensible to background traffic, those intervals (*i.e.*, with no user interaction) can be filtered out by looking at the pace at which activity windows are generated. Intuitively, when the user is active, multiple partitions are expected to be generated in a short time, while this effect is significantly reduced when only background traffic is present (see [36]).

Summary. Our results support the idea of identifying activity windows via passive measurements. We stress that the gradient policy is a heuristic, so not meant to be perfect. Its function is to enable us to focus on traffic dynamics and CPA knowing that the portion of the traffic under analysis is likely related to user engagement, hence meaningful to be dissected.

VI. Network Waterfall and Metrics

For each identified activity window, MCPA creates a download waterfall detailing traffic dynamics and performance.

Network waterfall. MCPA extracts transport (L4) and application (L7) per-flow metrics. At L4, it computes aggregated statistics (e.g., total duration, bytes, RTT), as well as protocol specific information (e.g., TCP, QUIC, FB0 handshake duration, IP addresses, ports). At L7, MCPA reports on HTTP transactions (e.g., metadata from request and response headers), TLS handshake (e.g., duration, if the handshake is full or fast, SNI, ALP protocols), DNS (e.g., domain name, CNAMEs, query resoution time). Moreover, each flow is split into *bursts* by grouping packets when interleaved by more than 2 RTTs. All the metrics are then represented as a download waterfall, a relevant visual aid to CPA (§VII).

Performance metrics. As discussed in §II, we consider AFT and SI suitable to study mobile apps traffic. However, we consider them only as baseline as we aim to avoid on-device screen recording. We are instead interested in studying the reliability of objective metrics based on passive traffic measurements. We define the *instant* metric *Transport Delivery Time* (TDT) as the time between the beginning of an activity window and the 95th percentile of the whole volume exchanged in the window.[7] We also define the equivalent *integral* metric *Transport Delivery Index* (TDI) as $\int_0^{TDT} 1 - x_B(t)dt$, where $x_B(t)$ is the percentage of total volume exchanged in the window up to time t. We highlight that TDI is similar to the *Object Index* introduced in [6] using TDT instead of PLT (recall that PLT does not apply for generic mobile apps §II). In the remainder of the section we investigate the penalties TDT and TDI introduce against the respective baselines AFT and SI. We consider also PLT as reference for browsing performance.

A. Evaluation

Web Browsing. Fig. 2(left) reports the Cumulative Distribution Function (CDF) of the deltas AFT-TDT and SI-TDI for alexa-T100 dataset. Both are well centered around zero, but TDI is a better proxy of SI than TDT is for AFT. Notice however that AFT-PLT presents a similar distribution as AFT-TDT. In other words, if PLT is the most popular metric to measure web performance, TDT is at least comparable. This is further corroborated considering PLT-TDT which presents a distribution well centered around zero.

Aggregate apps traffic. Fig. 2(right) reports the CDFs of AFT-TDT and SI-TDI deltas for both app-startup and app-click datasets. All curves are well centered around zero, but app-startup CDFs present a heavier negative tail. This resembles what was observed for browsing, *i.e.,* at startup more content is downloaded than what is required for the visualization, so TDT and TDI can over-estimate rendering deadlines. TDI is more sensible to this effect, while for 75% of the experiments TDT generates a $\pm$1.3s error at most.

Per-app traffic. To further investigate the deviations between instant metrics, Fig. 3 reports the deltas AFT-TDT as a function of TDT for each individual app. Considering app-startup (left plot), besides a few outliers, all apps present

[7] We experimented with other percentiles too (see [36]), but the 95th resulted the more robust to long tail effect (e.g., keep alive).

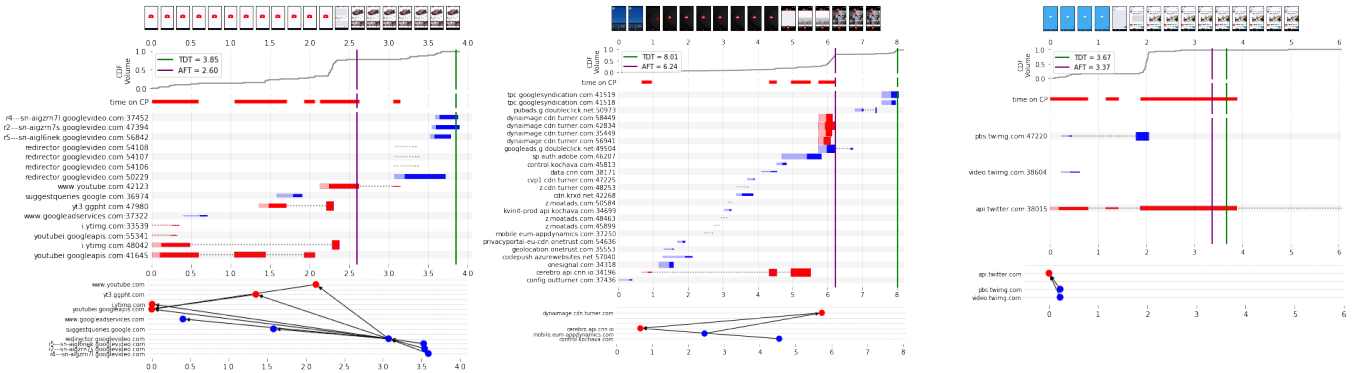

Fig. 4: Examples of download waterfall: YouTube (left); CNN (center); Twitter (right).

similar behavior, with variable deadlines in absolute scale, but TDT is triggered slightly after AFT as already observed in Fig.2(right). For app-click (right plot) errors are further reduced, with only Amazon showing larger penalties.

Summary. The analysis shows that metrics purely based on passive traffic monitoring are a reasonable approximation of AFT and SI, and at least as good as popular metrics such as PLT. This brings visibility on apps dynamics when AFT and SI cannot be measured, and more broadly they can significantly simplify QoE/performance analysis. There are clearly some corner cases and occasional outliers, as not all apps behave the same, but our analysis shows that TDT and TDI are reasonable heuristics to qualitatively capture delivery deadlines.

VII. Critical Path Analysis

CPA tools for browsing define the critical path based on a dependency graph capturing the relations between objects downloaded (§II-B). This graph is constructed "passively" exploiting the DOM built by the browser when rendering the webpage; however this technique is not applicable to generic mobile apps. Therefore, to discover critical traffic, MCPA uses an "active" approach based on traffic throttling. We use the `tc` utility to throttle one domain at a time to 1kb/s, and test the impact on the activity window delivery deadline. In particular, for each throttling scenario we perform 10 runs applying a p-value test (with 0.05 as significance level) to accept or reject the null hypothesis: a domain is critical if the deadline is always delayed across runs. Likewise, a similar test is applied to discover dependencies among domains (*i.e.,* by delaying domain A also domain B is delayed).

Overall, we define *Critical Set* (CS) as the set of domains impacting the delivery deadline, and we use it to create a dependency graph among domains. We define *Critical Path* (CP) as the whole set of flows generated by the CS. In other words, similarly to Lighthouse, MCPA CP is defined based only on network traffic, but it captures the whole traffic activities of a flow, rather than pinpointing specific objects/requests. It follows that the *time on CP* is the sum of time intervals where at least 1 critical flow is active. In the remainder of the section, we first present some examples of CPA on specific apps. Then, we discuss traffic properties across apps.

A. Dissecting individual apps traffic

Fig. 4 details the startup traffic dynamics for YouTube, CNN, and Twitter apps. For each app, we stack 6 views of the traffic: dependency graph, download waterfall, time on CP, CDF of the bytes exchanged, and a film strip showing the screen rendering progress. The dependency graphs show only domains having at least one dependency. In the download waterfall each row corresponds to a different flow (labeled with domain and destination port). Horizontal lines show bursts carried by flows (§VI), colored red if found critical (blue otherwise), while dotted lines indicate idle periods. Saturated colors reflect exchange of data, while pale ones correspond to DNS and handshakes (TCP, TLS, or QUIC). Finally, two vertical lines mark the AFT and TDT deadlines.

YouTube. Focusing on YouTube, the traffic before the AFT is almost entirely critical. This is composed of a mix of images (*i.ytimg.com* handles video thumbnails, while *yt3.ggpht.com* handles user related content such as avatars), control, and other structural elements of the app (e.g., fonts, javascripts). The download idle times hint to rendering cycles (fetch→process→render→iterate), as also confirmed by the film strip showing a "dummy" loading screen used to hide the actual rendering process. TDT is delayed due to video pre-fetching [26]. This is confirmed by app-click, where we observe the portion of video left being delivered on the already opened flows when the playback is triggered (see [36]).

CNN. Differently from YouTube, the majority of the traffic for the CNN app is not critical. After contacting *cerebro.api.cnn.io* (possibly a control domain), there are about 3s busy with only 3rd party and ads services communications, none of which is critical. Finally the control goes back to *cerebro.api.cnn.io* which triggers the rest of the critical traffic (*dynaimage.cdn.turner.com*). As for YouTube, rendering phases are possibly hidden by the loading screen, but more interesting is the macroscopic impact of 3rd party traffic which accounts for 55% of the overall deadline.

Twitter. The Twitter app instead has a very simple waterfall: only 3 flows, all twitter related, with only 1 being critical. We interpret this minimalist approach as an explicit design choice, but it would be interesting to know if applying content sharding and a few more flows could further reduce loading latency.

TABLE I: Critical path traffic characteristics.

	App-startup										App-click									
	fl.		dom.		vol.[kB]		TC[s]		TC break [%]		fl.		dom.		vol.[kB]		TC[s]		TC break [%]	
	abs	%	abs	%	abs	%	abs	%	dns	hshake	data	abs	%	abs	%	abs	%	abs	%	dns	hshake	data
Twitter	13	38	1	13	33	79	4	77	0	32	68	1	13	1	25	29	96	19	44	0	0	100
Facebook	7	40	2	40	836	97	9	61	1	6.3	92.7	2	40	2	67	1313	63	5	54	6	10	84
Instagram	16	56	2	25	1108	97	4	80	0	11.6	88.4	4	57	1	50	1538	90	2	50	0	0	100
Whatsapp	2	100	1	100	4	100	1	100	9	6	85	1	100	1	100	1	100	1	100	0	0	100
Snapchat	10	80	4	50	2802	91	11	70	3	23	74	2	22	1	33	194	75	6	17	2	7	91
Messenger	5	57	3	50	86	72	2	63	0	31.8	68.2	3	60	2	40	20	79	10	35	0	2	98
CNN	10	59	2	13	25	31	3	38	15	19.6	64.4	3	25	2	40	69	82	1	55	0	3	97
BBC	6	75	2	50	98	96	1	21	0	29	71	4	36	2	67	105	92	1	67	0	22	78
NewsBreak	27	66	5	25	152	92	5	63	0	20	80	19	43	7	78	96	20	3	73	0	13	87
Gmaps	17	65	6	46	870	99	4	57	0	37	63	3	60	2	100	870	98	2	52	0	0	100
Uber	18	81	6	43	238	95	13	53	0	25	75	3	43	1	50	13	11	5	85	0	0	100
Letgo	10	56	3	30	715	97	1	18	6	31	63	5	100	2	100	65	100	2	100	0	6	94
Amazon	33	67	5	45	1490	96	7	84	0.4	30.6	69	21	34	4	36	1650	92	12	80	0	6	94
Gmail	7	14	1	20	16	91	2	82	0	46.5	53.5	6	55	2	40	38	75	11	21	0	7	93
Outlook	4	57	2	50	20	91	2	79	3	32	65	5	83	3	75	9	100	0	35	0	75	25
Youtube	10	63	5	45	127	84	3	46	5.8	25	69.2	5	45	1	20	65	47	1	31	0	67	35
SoundCloud	10	43	2	20	715	99	8	76	0	84	78	1	17	1	33	120	99	1	44	0	0	100
Spotify	1	13	2	25	78	95	5	59	1	15	84	3	30	1	50	115	98	0	52	0	0	100
AVERAGE	11	57	3	38	523	89	5	63	2.5	24.8	72.7	5	48	2	56	351	79	4	55	0	12	88
Browsing	12	48	5	37	488	71	5.53	38	2.6	21.5	76											

B. Critical traffic properties across apps

Table I summarizes the critical traffic properties for both app-startup (left) and app-click (right). For each app we report the number of critical flows, domains, bytes both in absolute and percentage averaged across different runs. We also report the time spent on the critical path (TC) and how this is spent doing DNS, transport handshakes, and data transfers. Table rows are grouped by app categories.

Traffic volume. On average, 57% (48%) of flows, 89% (79%) of bytes are critical in app-startup (app-click). Differently from what we expected, in absolute scale the volume of bytes is still significant in app-click (351kB on average, almost 70% of the average volume in app-startup). Considering domains, 38% are critical in app-startup startup against 56% for app-click. There are macroscopic differences between apps, but no visible patterns within and between categories or scenarios. For instance, Whatsapp is an "outlier" as all traffic is carried over 1-2 flows, hence everything is critical. The only class that seems different is web browsing, which presents 48% (71%) of critical flows (bytes), -9% (-18%) with respect to apps startup.

Time on CP. For browsing also TC is lower, 38% against 63% (55%) in app-startup (app-click) as also detailed in Fig. 5. On the other hand, for both browsing and apps TC is similar in absolute scale (4-5s). In other words, despite the diversity in the actions triggered, results suggest that the differences in the critical traffic between startup and actual app usage could be less pronounced that one might think. As expected, data transfer has the largest impact on the critical path with 72.7% (88%) for app-startup (app-click). DNS is generally small except for a few cases. Conversely, protocol handshakes are heavier at startup (24.8% on average), but app-click shows unexpected bi-modal behaviour with either a heavy (e.g., 67% YouTube, 10% Facebook) or negligible weight.

Content type analysis. Extracting keywords from the domains, we split the traffic in 3 classes: *ad-hoc* (apps/websites specific domains), *cdn*, and *oth-serv* (e.g., 3rd party services, ad networks). We find that for apps (browsing) TC is split into 68% (33%), 25% (51%), and 7% (15%), while volume is split into 47% (25%), 52% (65%), and 1% (9%) for ad-hoc, cdn, and oth-serv respectively. In other words, apps network latency tends to gravitate towards app-specific domains. Those are not necessarily responsible only for control logic as they carry almost the same volume as CDNs. Conversely, browsing content is likely served by CDNs. Considering oth-serv, browsing spends 2× TC than apps, but downloads 9× more volume than apps.

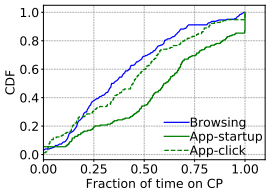

Fig. 5: MCPA time on CP for different types of traffic.

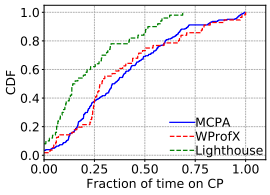

Fig. 6: Comparing network time on CP across CPA tools.

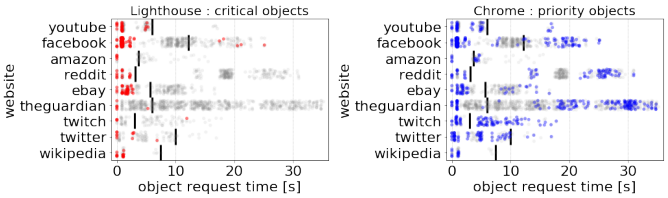

Fig. 7: Lighthouse critical path analysis.

VIII. Discussion

MCPA aims to identify critical traffic generated by generic mobile apps. A few other CPA tools for mobile apps have been presented, but none of them are applicable to our intent as they either require heavy on-device instrumentation or do not dissect traffic dynamics [32], [41]. However, restricting the focus to web browsing only, we can compare MCPA with WProfX (the WProf version for mobile browsing) and Google Lighthouse, both open sourced. Fig. 6 shows the CDF of the fraction of time on CP for the three tools. We highlight that for MCPA and Lighthouse, time on CP implicitly refers to only network activities, while WProfX reports also on parsing and rendering time, which we exclude for the comparison.

WProfX profiles the impact of webpages loading activities on PLT. Notice the strong similarity of MCPA and WProfX CDFs, with both tools reporting 38% of time on CP on average. This implies that MCPA, even if based on traffic analysis only, is comparable with an in-browser profiling engine.

Lighthouse reports the webpage *Critical Request Chains* (CRCs) pinpointing to objects generating bottlenecks.[8] As visible in Fig. 6, Lighthouse reports a shorter time on CP than both WProfX and MCPA. We found that MCPA generally classifies a few more domains as critical than Lighthouse (details reported in [36]), but the same is true for WProfX too. The reason of the discrepancy resulted clear only by investigating Lighthouse source code, *i.e.,* it is due to an internal design choice not publicly documented. Specifically, Lighthouse marks objects as critical if they have a *network priority* higher than medium (*i.e.,* the browser schedules objects fetch early on), and they are neither images, XML HTTP

[8] https://developers.google.com/web/tools/lighthouse/audits/critical-request-chains

Request (XHR), nor server push(ed) content. This results in a "constrained" view of the traffic as reported for a subset of websites by the strip-plots in Fig. 7: grey dots represent all requests; red dots (left plot) mark critical objects; blue dots (right plot) marks prioritized objects; vertical black lines mark the AFT. Notice how Lighthouse is biased towards the first part of the download, which possibly involves only "structural" properties of the webpage rather than actual content.

Beside the fine-grained details, the tools comparison highlights a more subtle problem: the lack of standard methodologies to pinpoint what is critical, and how to perform root cause analysis related to those bottlenecks. These goals go beyond the purpose of our work, which instead addresses a prior and more fundamental requirement: to ease the study of generic mobile apps. We demonstrated that network measurements can be effective and easier to adopt than rendering based metrics such as AFT/SI. Moreover, our definition of critical path aims to discover any critical network activity without any restriction on the type, so to capture traffic dynamics as a whole. To test MCPA we adopted the standard practice of an instrumented device, with the intention to demonstrate that this might not be necessary. This can open the doors to a new class of tools easier to deploy than current state of the art techniques, without significantly sacrificing accuracy. In this way, app developers and mobile operators could better dissect traffic dynamics (e.g., TCP/TLS handshake, TCP fast open [31], app-specific protocols, control logic, or pre-fetching) by means of at-scale measurement campaigns.

REFERENCES

[1] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin, "Flywheel: Google's data compression proxy for the mobile web," in *Proc. USENIX NSDI*, May 2015.

[2] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements," in *Proc. ACM HotMobile*, 2014.

[3] M. Almeida, M. Bilal, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Varvello, and J. Blackburn, "Chimp: Crowdsourcing human inputs for mobile phones," in *Proc. WWW*, 2018.

[4] M. Almeida, A. Finamore, D. Perino, N. Vallina-Rodriguez, and M. Varvello, "Dissecting dns stakeholders in mobile networks," in *Proc. ACM CoNEXT*, 2017.

[5] F. A. Blog, "U.s. consumers time-spent on mobile crosses 5 hours a day," 2017, http://flurrymobile.tumblr.com/post/157921590345/us-consumers-time-spent-on-mobile-crosses-5.

[6] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the quality of experience of web users," in *Proc. SIGCOMM Internet-QoE*, 2016.

[7] J. Brutlag, Z. Abrams, and P. Meenan, "Above the fold time: Measuring web page performance visually," https://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692, 2011.

[8] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "Klotski: Reprioritizing web content to improve user experience on mobile devices," in *Proc. USENIX NSDI*, 2015.

[9] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau, "Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis," in *Proc. ACM IMC*, 2014.

[10] D. Collins, "Mobile conversion rates lag behind desktop," 2017, https://grafik.agency/insight/mobile-conversion-rates/.

[11] S. Colwyn, "New consumer media consumption research," 2014, https://www.smartinsights.com/marketplace-analysis/customer-analysis/consumer-media-device-use/.

[12] D. N. da Hora, A. Alemnew, C. Vassilis, R. Teixeira, and D. Rossi, "Narrowing the gap between qos metrics and web qoe using above-the-fold metrics," in *Proc. PAM*, 2018.

[13] J. Erman, V. Gopalakrishnan, R. Jana, and K. Ramakrishnan, "Towards a SPDYier Mobile Web?" in *Proc. ACM CoNEXT*, 2013.

[14] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the ACM IMC*, 2010.

[15] Q. Gao, P. Dey, and P. Ahammad, "Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold qoe," in *Proc. SIGCOMM Internet-QOE*, 2017.

[16] U. Goel, M. Steiner, M. P. Wittie, M. Flack, and S. Ludin, "Detecting cellular middleboxes using passive measurement techniques," in *Proc. PAM*, 2016.

[17] Google, "Speed Index," https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index, 2008.

[18] Google, https://developers.google.com/web/tools/lighthouse/, 2017.

[19] B. Han, F. Qian, S. Hao, and L. Ji, "An anatomy of mobile web performance over multipath tcp," in *Proc. ACM CoNEXT*, 2015.

[20] B. Han, F. Qian, and L. Ji, "When should we surf the mobile web using both wifi and cellular?" in *Proc. Workshop on All Things Cellular*, ser. ATC '16.

[21] T. Hofeld, F. Metzger, and D. Rossi, "Speed index: Relating the industrial standard for user perceived web performance to web qoe," in *QoMEX*, 2018.

[22] C. Kelton, J. Ryoo, A. Balasubramanian, and S. R. Das, "Improving user perceived page load time using gaze," in *Proc. USENIX NSDI*, 2017.

[23] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopolou, "AntMonitor: A System for Monitoring from Mobile Devices," in *ACM C2B1D*, 2015.

[24] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y.-M. Wang, "Webprophet: Automating performance prediction for web services," in *Proc. USENIX NSDI*, 2010.

[25] Y. Ma, X. Liu, S. Zhang, R. Xiang, Y. Liu, and T. Xie, "Measurement and analysis of mobile web cache performance," in *Proc. WWW*, 2015.

[26] R. Mok, V. Bajpai, A. Dhamdhere, and k. claffy, "Revealing the Load-balancing Behavior of YouTube Traffic on Interdomain Links," in *Proc. PAM*, 2018.

[27] Monetate, "Benchmarks and research - eq1," 2018, https://info.monetate.com/rs/092-TQN-434/images/EQ1-2018_First-Impressions.pdf.

[28] D. Naboulsi, M. Fiore, S. R., and R. S., "Large-scale mobile traffic analysis: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 124–161, Oct. 2015.

[29] J. Nejati and A. Balasubramanian, "An in-depth study of mobile browser performance," in *Proc. WWW*, ser. WWW '16, 2016.

[30] F. Qian, S. Sen, and O. Spatscheck, "Characterizing resource usage for mobile web browsing," in *Proc. ACM MobiSys*, 2014.

[31] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "Tcp fast open," in *Proc. ACM CoNEXT*, 2011.

[32] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayandeh, "Appinsight: Mobile app performance monitoring in the wild," in *Proc. OSDI*, 2012.

[33] J. P. Rula and F. E. Bustamante, "Behind the curtain: Cellular dns and content replica selection," in *Proc. ACM IMC*, 2014.

[34] Sandvine, "Global Internet Phenomena," https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf, 2018.

[35] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: Smartphone fingerprinting via application behaviour," in *Proc. ACM WiSec*, 2013.

[36] G. Tangari, A. Finamore, D. Perino, M. Charalambides, and G. Pavlou, "Technical report - Tackling Mobile Traffic Critical Path Analysis With Passive and Active Measurements," https://www.dropbox.com/sh/rk853z5e49l1mjy/AADOvq19EQ05R5ZDK_Jt0w1Ya?dl=0, 2019.

[37] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki, "EYEORG: a platform for crowdsourcing web quality of experience measurements," in *CONEXT*, 2016.

[38] W3C, "Navigation timing level 2," 2018, https://w3c.github.io/navigation-timing/\#introduction.

[39] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *Proc. USENIX NSDI*, 2013.

[40] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up web page loads with shandian," in *Proc. USENIX NSDI*, 2016.

[41] L. Zhang, D. R. Bild, R. P. Dick, Z. M. Mao, and P. Dinda, "Panappticon: Event-based tracing to measure mobile application and platform performance," in *Proc. CODES+ISSS*, 2013.