A Worldwide Look Into Mobile Access Networks Through the Eyes of *AmiGos*

Matteo Varvello[†]
[†]Nokia Bell Labs
Holmdel, USA

Yasir Zaki[‡]
[‡]NYU Abu Dhabi
Abu Dhabi, UAE

***Abstract*—How does the mobile experience compare between Germany and Nigeria?** **There is currently no public data or test-bed to provide an answer to this question. This is because deploying and maintaining such test-bed can be both challenging and expensive. To fill this gap, this paper proposes a novel test-bed design called "`AmiGo`", which relies on travelers carrying mobile phones to act as vantage points and collect data on mobile network performance. The `AmiGo` design has three key advantages: it is easy to deploy, has realistic user mobility, and runs on real Android devices. We further developed a suite of measurement tools for `AmiGo` to perform network measurements, e.g., `pings`, speedtests, and webpage loads. We leverage these tools to measure the performance of 24 mobile networks across five continents over a month via an `AmiGo` deployment involving 31 students. We find that 50% of networks face a 40-70% chance of providing low data rates, only 20% achieve low latencies, and networks in Asia, Central/South America, and Africa have significantly higher CDN download times than in Europe. Most news websites load slowly, while YouTube performs well. We made both test-bed and measurement tools open source.**

***Index Terms*—mobile, performance, testbed**

I. Introduction

Mobile traffic has been on the rise, even surpassing its desktop counterpart. Data from Google Analytics' Benchmarking [20] shows that mobile devices drove 61% of visits to U.S. websites in 2020, up from 57% in 2019. Desktops were only responsible for 35.7% of all visits in 2020, and tablets drove the remaining 3.3% of visitors. Globally, 68.1% of all website visits in 2020 came from mobile devices.

The research community has investigated how to improve the performance of mobile users, e.g., by improving how fast pages load [11]–[13], [22], [28], [32], [36], [45], or reducing the data consumption [6], [29]. These works are motivated by the *challenging* conditions which mobile users face, such as low bandwidth or high network latencies. However, a comprehensive and open source measurement study of mobile operators in the wild is still lacking, as previous work [25], [39] only focused on few mobile operators, mostly in the US and Europe, or few applications.

Performing such study is challenging due to the lack of a large-scale test-bed in the wild. Even the popular MONROE [39] currently only spans 11 mobile networks in 4 European countries; further, its vantage points are not real mobile devices and have limited mobility. Our first contribution is a novel test-bed design, named `AmiGo`, which tackles these limitations. The novelty of the `AmiGo` design lies in the idea to leverage *friends* (hence the name) to carry – not use – mobile phones while travelling, guaranteeing network connectivity, *i.e.,* WiFi and/or mobile data when possible. We have open sourced all code behind `AmiGo` [35] to help measurement researchers deploying their own test-beds in the wild. In the following, we summarize the main contributions of this paper.

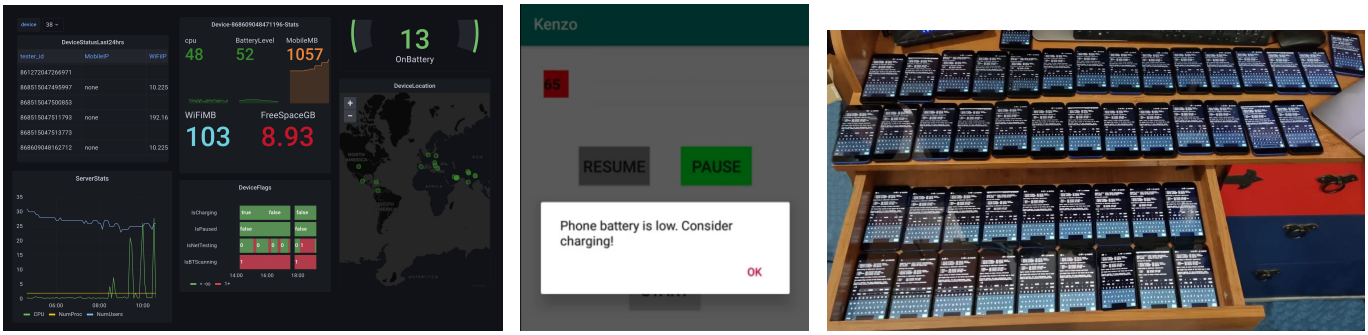
978-3-903176-58-4 ©2023 IFIP

The `AmiGo` test-bed: It consists of *measurement endpoints*, low-end Android mobile phones (Redmi Go [8]), and a *control server*. These Android devices are rooted and equipped with `termux` [43] – an Android terminal emulator and Linux environment – thus allowing fine-grained instrumentation and data collection. The phones frequently report to the control server (running in the cloud) their current status, e.g., connectivity (WiFi or mobile) and battery level. The controller monitors the phones' health, schedule experiments, and collect data. Finally, a mobile application allows to interact with the "amigos", e.g., to pause an experiment or show notifications.

Full stack networking experiments: We have developed and open sourced [35] several tools to perform measurements via `termux`: speed-tests, traceroutes, DNS queries, HTTP GET of popular CDN objects, webpage loads, and YouTube tests. By combining these tools, we measure multiple metrics *across the stack*, e.g., mobile bandwidth, latency to DNS services and content providers, up to user experience when browsing or watching videos.

Measurement campaign: We recruited 31 university students to participate to (an instance of) the `AmiGo` test-bed while traveling back to 24 different countries during the 2021 winter break (with three countries having more than one student). Overall, this allowed us to collect data across 24 mobile networks. Our key findings are as follows:

- *Slow* mobile operators – download speed wise – tend to be consistently slow. In contrast, *fast* mobile operators are less consistent. It follows that, even with fast operators, applications that require high bandwidth may suffer from frequent slowdowns.
- *Exceptional* latency (<20ms) is limited to few mobile networks (less than 20%) and content providers (Cloudflare and Google). Amazon is most likely to suffer from *less desirable* latencies (>150ms) and highly correlated with long network path, suggesting a smaller content distribution network.
- Apart from few African operators, DNS resolution on mobile mostly requires less than 100ms. Several operators (e.g., Telenor in Pakistan, IND Airtel in India, and Telcel in



(a) Graphana dashboard. (b) Android application. (c) Automated preparation of MEs.

Fig. 1. Several components of the AmiGo test-bed.

Mexico) sometimes rely on Google DNS, we conjecture in presence of network disruptions.

- News websites are often too *heavy* for low-end mobile devices, especially in mobile networks from developing regions. Given the importance of a diversified news outlet, they should prioritize optimizing their website to provide a better user experience for all users, regardless of their device or location. In contrast, YouTube shows better performance across the board.

II. Test-bed Architecture

This section describes the AmiGo testbed, which consists of a *control server* to remotely manage mobile *measurement endpoints* (MEs). Our design is generic and can be re-used to build in-house testbeds which aim at controlling multiple devices deployed in the wild. We have thus opened source our code [35] to help measurement researchers deploying similar testbeds in the wild.

A. Control Server

The control server has three main tasks. First, it monitors the status (battery level, GPS location, etc.) of the MEs. Second, it instruments MEs with *automation instructions* or new commands/actions which are not part of their default behavior, e.g., open a reverse SSH tunnel to enable debugging. Third, it maintains a dashboard visualizing the current status of devices and ongoing experiments (see Figure 1(a)).

The control server is implemented in Python and provides restful APIs which the MEs call to: 1) report their current status (e.g., battery level and connectivity), and 2) retrieve *instrumentation code*. The Python code also maintains a `postgres` [40] database which stores both device status updates and instrumentation code, *i.e.,* `shell` commands to be executed in `termux` (see below). Finally, `Graphana` [33] is used to build a visual dashboard allowing to identify potential issues with test-bed and/or experiments.

B. Measurement Endpoint (ME)

Rationale and Overview: Our rationale is to use a real mobile device as a ME. Mobile devices are easy to carry, thus enabling experiments across a plethora of networks and realistic conditions. They support multiple access networks, which allows flexibility for both remote instrumentation (e.g., via WiFi) and experiments (e.g., WiFi or mobile, even switching between 3G and 4G). Mobile devices also support a wide range of experiments, e.g., from `ping` to Web browsing. Finally, the data collected via a real device is representative of what experienced by actual users in the wild.

Name	Description
vrsNum	code version number
timestamp	epoch timestamp at time of report
uid	unique device identifier
airplaneMode	status of airplane mode
googleStatus	status of Google authorization
uptime	how long has the device been running
isPaused	whether the user pause our mobile app
freeSpaceGB	available space on device
cpuUtilPer	current CPU utilization
memInfo	available memory
batteryLevel	percentage of battery available
isCharging	whether the device is charging or not
gpsLoc	current GPS location
networkLoc	current network-provided location
foregroundApp	current app in the foreground
isNetTesting	status of network measurements
wifiIP	WiFi IP address
wifiSSID	SSID of WiFi network connected to
wifiQual	quality of WiFi network signal
todayWiFiData	data used on WiFi for the day
mobileIP	mobile IP address
mobileSignal	quality of mobile network signal
todayMobileData	data used on mobile for the day

TABLE I
Summary of data reported by a measurement endpoint to the control server with a 5 min frequency.

We chose a `Redmi-Go` [8] as `AmiGo`'s ME. This device is based on Android which allows full instrumentation while being the most popular operating system in the world [16]. It is also a cheap device (retail price of $70), which is paramount when the goal is to realize a test-bed with a large number of MEs. Note that despite being a low-end device (1.4 GHz quad-core CPU and 1 GB of RAM) our benchmark shows little to no impact to most measurements (with the exception of webpage loads, see Section V). Third, it can be easily rooted, allowing full control via `termux` [43], an Android terminal emulator and Linux environment. This gives us the flexibility to write

any instrumentation, from low level utilities like `ping` to app automation, and data collection, even when root privilege is needed. We later discuss how this extends to other devices.

Design and Implementation: We refer to the core code of the ME as `AmiGo`'s *agent*. The agent has overall three key tasks. First, it monitors the endpoint resources, e.g., battery level and connectivity. Second, it interacts with the control server, either to report its state (see Table I) or to collect instrumentation for new experiments to run. We leverage a combination of classic Linux tool (e.g., `ifconfig`), Android tool (e.g., `dumpsys`), and termux tools (e.g., `termux-location`) to populate the state information which is reported to the control server every 5 minutes. Third, it controls when to run a list of pre-installed experiments by matching a set of rules, e.g., only run with a given frequency when on mobile. Each experiment consists of a `shell` script, thus allowing for easy customization of the ME's behavior (see Section III-A).

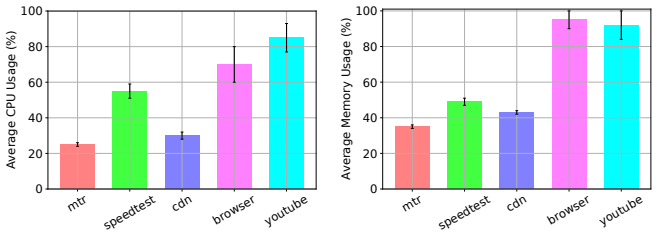
We use `cron` to ensure the agent is started at each phone reboot, e.g., in case the device running out of battery. Further, we use it to force a device reboot each night to "clean" potential wrong states reached during any of the experiments. We use the `Boot Apps` Android app [18] to ensure `termux` is launched at reboot, which in turns enable `cron` and `sshd`.

Each device is instrumented with a private SSH key used to: 1) pull code updates from `github` [5], 2) open a reverse tunnel to the control server, if requested to do so, which is useful for debugging misbehaving MEs. To guarantee a consistent state across all devices – and speedup the test-bed preparion job – the installation of every app and termux package is automated and relies on local packages provided by APKMirror [7]. Figure 1(c) shows the automated preparation of multiple MEs.

Mobile Application: It acts as `AmiGo`'s GUI and accomplishes three tasks. First, it shows the device identifier for simple communication between test-bed maintainers and a volunteer carrying the ME in presence of a concern. Second, it allows to pause an ongoing experiment. This is required when the volunteer needs to interact with the device, e.g., to connect to a WiFi network. Without this feature, any ongoing experiment could collide with the user causing potential issues. Finally, it notifies the user when the mobile phone requires charging (see Figure 1(b)), which is often neglected by volunteers (see Section VII).

Google Account: Android devices require a Google account. We have contacted Google asking for a testing account, with no luck. We have thus setup each device with the same Google account, given that no limitation on the number of devices exists. This has triggered random requests to verify our account, by entering its username and password. Fortunately, this operation can be automated by detecting the presence of the `MinuteMaidActivity` when launching YouTube.

Limitations: The main limitation of the `AmiGo` agent is that it currently targets a specific Android phone: Redmi Go. We purposely focused on one device to enable fair comparison of results across different mobile operators. Still, it would be interesting to extend to more recent and powerful devices, e.g., to investigate 5G connectivity. Extending to other Android devices is simple – as long as they can be rooted – as `AmiGo`'s agent only relies on low level instructions shared among Android devices. Extending to iOS would also be of interest although hard due to its limited automation capabilites.



(a) CPU utilization per test. (b) Memory utilization per test.

Fig. 2. Benchmarking of CPU and memory usage at the ME (`Redmi-Go` with 1.4 GHz quad-core CPU and 1 GB of RAM). Each bar reports the average CPU/memory usage per networking test. Errorbars report standard deviation.

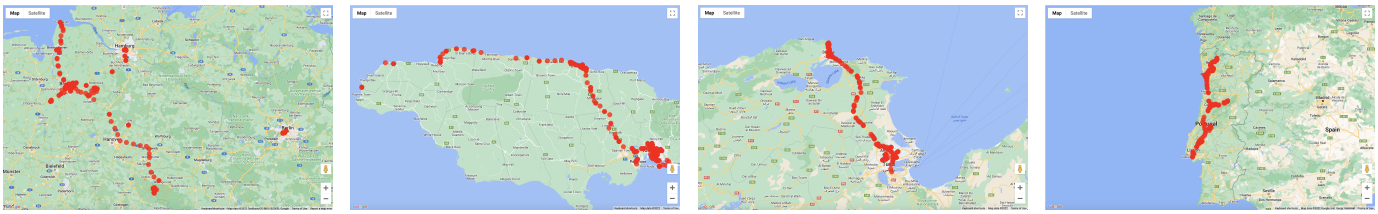
A potential additional issue with the device selection is that it might have an impact on the measurements performed, due to its limited hardware (1.4 GHz quad-core CPU and 1 GB of RAM). We have benchmarked CPU (Figure 2(a)) and memory (Figure 2(b)) usage when performing the measurements described in Section III-A. CPU-wise, most experiments are not concerning given that the device's CPU is rarely under stress, only in some cases YouTube approaches 90% CPU usage. The same is not true for memory usage which is instead fully occupied for both browser and YouTube experiments. This implies that higher level experiments can be partially impacted by the device we chose. We acknowledge this as a potential limitation due to the need to keep the cost per device low, and reach a large number of mobile networks.

III. Data Collection

This section describes the experiments we devised to measure the performance of mobile networks using the `AmiGo` test-bed. We instrument the agent to schedule experiments every 30 minutes when the ME is *only* connected to a mobile network. Note that we asked volunteers to connect the ME to their home WiFi as a simple mean to avoid running most experiments from their home, where we expect them to spend the majority of the time. We further monitor data consumption to avoid consuming more than 4 GB per day. As for the test-bed code, we have also open sourced it for each experiment.

A. Experiments Description

Access Characteristics: Mobile networks are characterized by three main metrics: download/upload speed, latency, and loss probability. We use a combination of tools to compute each metric. First, we measure download/upload speeds using *Speedtest CLI* [38], a Linux-native Speedtest tool backed by Ookla®.Next, we derive network losses from pcap files captured via `tcpdump` while loading popular webpages. Finally,



(a) Germany (b) Jamaica (c) Tunisia (d) Portugal

Fig. 3. Visualization of AmiGo volunteers mobility in four countries: Germany, Jamaica, Tunisia, and Portugal.

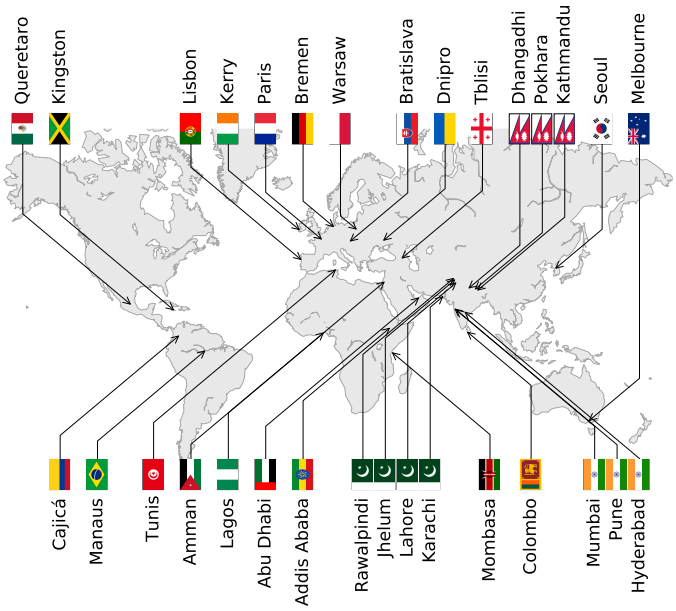


Fig. 4. Geographical distribution of the AmiGo test-bed deployment between December 2021 and January 2022.

we use mtr [30] – a network diagnostic tool combining ping and traceroute – to derive latency and network path towards popular content providers (Amazon, Facebook, Google, YouTube) and DNS providers (Google and Cloudflare). The rationale, as suggested in [17], [50], is that these providers employ edge nodes which are commonly close to the users.

DNS Performance: DNS is a critical component of every mobile application. We instrument our MEs to use the DNS provided by the mobile network they are connected to. We evaluate DNS on mobile using statistics extracted from pcap files of webpage loads.

CDN Performance: A Content Delivery Network (CDN) is a popular networking tool used to accelerate content retrieval, thus improving the user experience, by moving popular content close to a user's network location. We have identified a popular JS file (jquery.min.js version 3.6.0, or the last version at the time of our measurement campaign) which is hosted at multiple CDNs (Cloudflare, Facebook CDN, Google CDN, Highwinds CDN, jsDelivr, and Microsoft Ajax CDN). Note that jsDelivr advertises optimal performance by matching each request to an "optimal" CDN based on uptime and performance [27].

We iterate through these CDNs fetching jquery.min.js using cURL instrumented to report the total download time. We further collect HTTP headers since, for some CDNs, they report whether the file was found in a cache, or not. Specifically, Cloudflare, jsDelivr, and Microsoft Ajax report cache HITs or $MISS$es using two HTTP response header fields: x-cache and cf-cache-status (from Cloudflare [15]). Fastly also use the x-cache header to report (e.g., '$HIT, MISS$', '$MISS, HIT$') *where* a cache hit or miss has occurred: either at the edge (second entry) or at the shield (first entry). A shield is a mid-tier caching layer between origin server(s) and edge servers [21]. Since jsDelivr relies on a network of CDNs, including Fastly, in some cases it also reports where a cache hit/miss has occurred.

Application Performance: Web browsing and video streaming are two popular mobile applications which are *easy* to automate, differently from even more popular apps like TikTok or Instagram. For Web measurements, we load several news websites via Google Chrome, namely: cnn.com, wsj.com, bbc.com, foxnews.com, and washingtonpost.com. During a page load we record pcap traces and a video which is then fed to visualmetrics [49] to extract performance timing metrics such as SpeedIndex [47] – which reports the time it takes for the visible parts of a webpage to be displayed.

For video streaming, we automate YouTube. This automation requires interacting with its GUI and is thus specific to the Redmi Go, or better the version of YouTube under test (17.43.46) and a device with a screen resolution of 720x1280 pixels. Extending to other devices is straightforward, given the logic of the automation does not change, but it requires some manual verification. YouTube allows to enable "stats-for-nerds" [24] which report information like buffer occupancy and number of lost frames. Such information is reported on screen, over the video, and can then be copied on the clipboard and dumped to a file. We also collect pcap traces while performing YouTube experiments. We used a (up to) 4K video specifically produced for testing (*https://www.youtube.com/watch?v=TSZxxqHoLzE*).

B. Data Overview

World Coverage: We have deployed AmiGo MEs via 31 university students while travelling back home during the

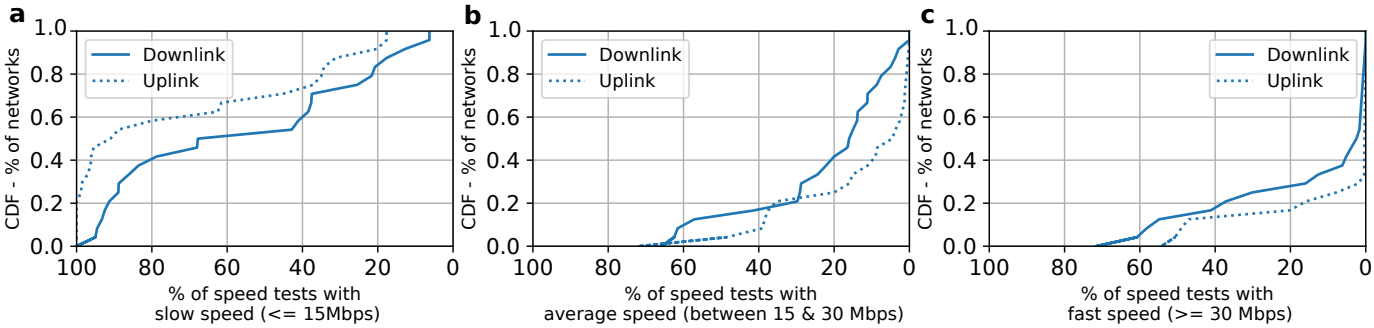


Fig. 5. Speed tests' measurements results showing the CDF of percentage of mobile networks with downlink and uplink speeds quantified as: (a) slow (<=15Mbps), (b) average (between 15 and 30 Mbps), (c) fast (>=30Mbps).

2021 winter break. The students were recruited based on the home countries they were traveling back to, as well as their willingness to carry the phones as much as possible. The data collected spans 31 cities in 24 countries (see Figure 4) between December 15th, 2021 and January 30th, 2022. Multiple students travelled to different cities in three countries: 4 to Pakistan, 3 to India, and 3 to Nepal. The students were instructed to purchase a mobile SIM card from their destination with a data plan around 40/50 GB. The choice of the mobile network operator was left to the students based on their own convenience. The students did move within each country, sometimes even going to multiple cities. Figure 3 shows sample users' movements in four different countries.

Mobile Access Type: Overall, the vast majority of the measurements were performed on 4G: 60% of the phones had 90% of their measurements performed on 4G. Nevertheless, 10% of the phones never encountered a 4G mobile network during their tests. The remainder 30% of phones had 40% - 90% of their tests performed on 4G.

User Mobility: We use GPS data to monitor user mobility. Most users were quite mobile, exploring more than 200 km during the measurement campaign and spending about 100 hrs outdoor (roughly 4.5 days). Some users ventured in trips; for example, one user went from France to Spain, resulting in being located 700 km away from what was marked as the home location. In few cases, users spent very little time outside due to safety restrictions caused by the rapid spread of Omicron (COVID-19 variant) [48].

Data Usage: Data consumption depends on two factors: 1) how much users move, 2) the 4 GB limit per day we set. The rationale of the latter limit was not to consume more than 40/50 GB (the target data plan we recommended acquiring) over a 10/15 days period (the average expected travel duration), so to avoid the inconvenience to recharge the data plan. Overall, our daily limit was very conservative since the median total data consumption was close to 6 GB. Few participants (Poland and Germany) consumed a lot of mobile data: 44.9 GB (over 44 days) and 26 GB (over 29 days), respectively. This was driven by high user activity and increased trip duration due to COVID-related travel restrictions.

C. Ethics

Given that we recruited participants to carry instrumented mobile devices (`AmiGo` MEs), we obtained an institutional review board (IRB) approval (HRPP-2021-185) to conduct these studies. In addition, one of the authors have completed the required research ethics and compliance training, and was CITI [4] certified. Participants were also provided with a consent form to read, and sign, acknowledging their willingness to participate. Further, they were given the opportunity to ask questions about the study and what data was being collected.

We asked participants to carry `AmiGo` MEs, charge them, install a mobile data sim-card, and connect them to WiFi when possible. We also instructed the participants not to use the mobile phones or add any of their personal information or logins. As such, we do not collect any unidentifiable, sensitive, or personal information about the participants. The only foreseeable concern that might put our users' privacy at risk is the collection of the phones' GPS data, which in principle can reveal the participants movements. We informed the participants beforehand about this concern and we obtained their written consent that they approve this collection.

IV. Network Characterizations

This section analyzes low-level networking experiments, e.g., speedtests, traceroutes, and DNS lookups. For a given metric, our data-set contains multiple data points for each mobile operator. Instead of plotting one Cumulative Distribution Function (CDF) per mobile operator, to ease visibility we plot the CDF of the percentage of mobile networks for which a target metric is within a range. This analysis, inspired by Chrome User Experience Report (CRuX) [23], allows to comment on the probability, expressed in number of tests, that a certain condition is met at a fraction of the mobile networks.

Speed Tests: Figure 5 shows the CDF of the percentage of mobile networks that have *slow*, *average*, and *fast* downlinks (solid) or uplinks (dashed). A slow downlink/uplink is characterized by a speed smaller than 15 Mbps; average refers to a speed between 15 and 30 Mbps. Finally, fast refers to a speed higher than 30 Mbps. These thresholds are drived form the

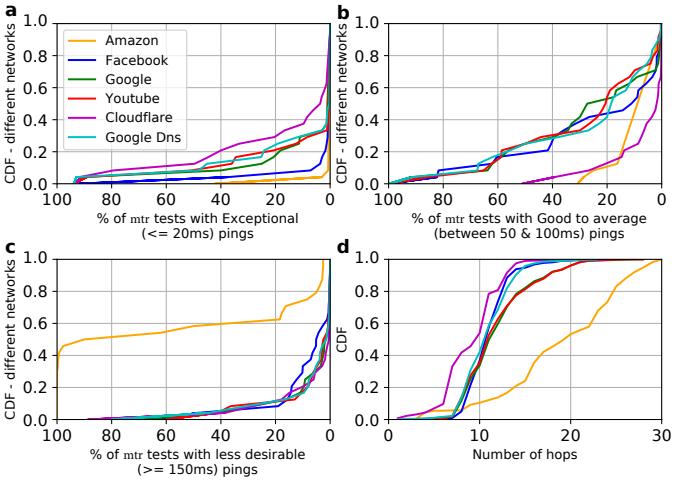


Fig. 6. CDFs of percentage of mobile networks with latencies per content provider classified as: (a) exceptional ($\leq 20ms$), (b) good to average (between $50ms$ and $100ms$), (c) less desirable ($\geq 150ms$). Figure (d) shows CDFs of the number of hops between MEs and each content provider. Latency and number of hops results are obtained using `mtr`.

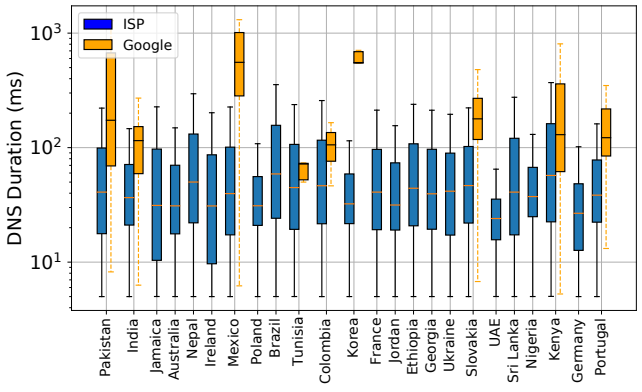


Fig. 7. Box plots of DNS lookup times per mobile operator, comparing when the phones used local DNS servers (provided by the mobile operator) versus Google DNS, if available. Note that Google DNS was not setup on the phone, but likely used by the mobile operator in presence of outages or high load.

SpeedTest Global Index which ranks 140 operators by their upload/download speeds [37].

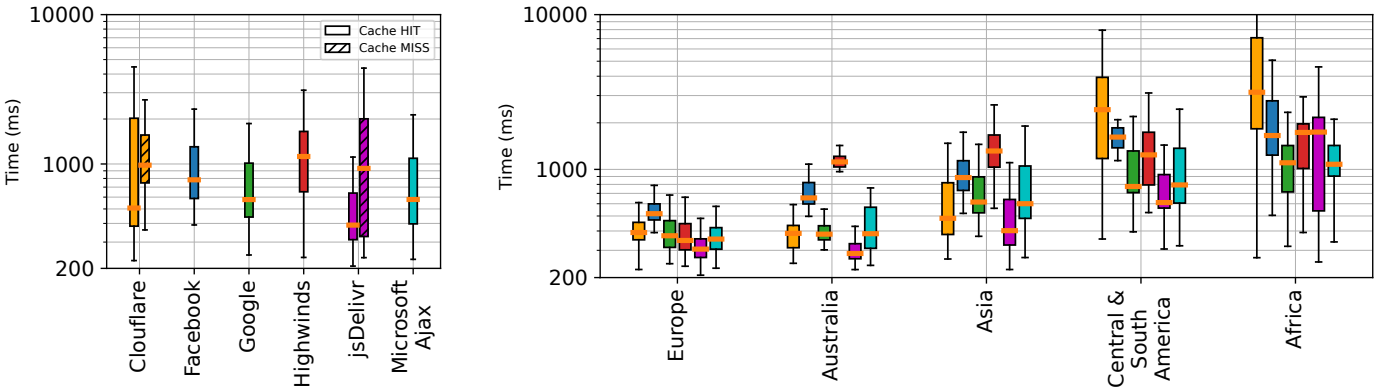
As also reported in [37], Figure 5 shows that slow speeds are more likely on uplink than on downlink, e.g., 90% of the uploads were slow for 50% of the mobile networks (whereas such frequent slow downloads only happen for 25% of the mobile networks). Average and fast speeds are instead more likely in download than upload. For both upload and download speeds, slow tests are more *consistent*, meaning that a slow mobile operator tends to be slow for the majority of the test, e.g., 40% of the operators have constant speed lower than 15 Mbps for 80-100% of the tests (e.g., Flow in Jamaica). Conversely, fast mobile operators are less consistent, *i.e.,* most have less than 50% of the measurements which can be considered fast (e.g., Telcel in Mexico). This result suggests that applications which constantly require high bandwidth, e.g., volumetric videos or virtual/augmented reality, might suffer from frequent slowdowns.

Mtr Pings and Traceroute: Figure 6 (a), (b), and (c) show respectively the aggregated results based on the quality of the mobile network latency tests [19]: 1) *exceptional* ping which falls below 20 ms, 2) *good* to *average* ping between 50 and 100 ms, and 3) *less desirable* ping which exceeds 150 ms. These figures also show the latency CDFs split based on the tested domain: `Amazon`, `Facebook`, `Google`, `YouTube`, `Cloudflare DNS`, and `Google DNS`.

Figure 6 (a) shows that exceptional latencies are rare. Only a small percentage of mobile networks (*i.e.,* 20%) have exceptional latencies to about four of the tested servers: `Cloudflare` (40% of tests), and `Google` products (DNS, YouTube, and search engine) with 20-25% of the tests. Conversely, both `Facebook` and `Amazon` have nearly no tests with exceptional latencies. For the "good to average" pings, apart from `Cloudflare` and `Amazon`, Figure 6 (b) shows that latency to all other popular providers are similarly split between mobile networks and number of tests. For example, 20% of mobile networks have about 60% of their tests ranked as "good to average"; this suggests that such ping values are mostly due to network conditions at the mobile networks rather than provider performance. `Amazon` and `Cloudflare` are outliers due to their limited number of samples in this category: indeed, the majority of their samples fall under exceptional pings (in the case of `Cloudflare`) and less desirable (in the case of `Amazon`), for which 50% of mobile networks have 80% less desirable pings. To better understand the previous results, we investigate the length of the network path between mobile users and each measured service. Figure 6 (d) shows that `Cloudflare` and `Amazon` have the shortest and longest number of hops to our mobile users, thus confirming their latency results observed above.

Domain Name System: Figure 7 shows box plots of DNS lookup times—extracted from Web browsing tests— per mobile operator. Blue box plots refer to lookup times obtained when a *local* (provided by the mobile operator) DNS is used; orange box plots refer to lookup times obtained via Google DNS—identified by `8.8.8.8` or `8.8.4.4` Mobile phones were instrumented to receive the DNS configuration from their operators; it follows that several operators (e.g., Telenor in Pakistan, IND Airtel in India, and Telcel in Mexico) sometimes rely on Google DNS. Note that this event is quite rare (less than 1% of the measurements) with the exception of Telkom (Kenya), where Google DNS was reported in 27% of the measurements.

Figure 7 shows that the median duration of DNS lookups with operator-provided DNS is quite fast, about 40ms. This is likely due to the high popularity of the websites under test (see Section III-A), which can be resolved from the DNS cache. When comparing mobile operators, their DNS achieve overall similar performance. Conversely, Google DNS lookups are much slower (up to a 10x increase for Telcel in Mexico). We conjecture that Google DNS is used as a backup solution



(a) CDN download time as a function of cache HIT or MISS.

(b) CDN download time per continent.

Fig. 8. Box plots of total download time for six popular CDN servers: (a) overall comparison of the download time for each CDN, including the impact of cache MISSes (for `Cloudflare` and `jsDelivr`), (b) comparison of the download time split per five continent for each CDN server.

in presence of outages or high load, since its usage was rare in our experiments.

Content Delivery Networks: We study the performance of each mobile operator towards six popular CDNs by downloading the last version of `jquery.min.js` hosted by each of them. On average, this file was downloaded about 117 times per CDN; all downloads were carried out over HTTP/2 with TLSv1.3, using various cipher suites.

We start by comparing the performance of each CDN globally, *i.e.,* aggregating measurements from 31 mobile networks (see Figure 8(a)). When possible (`Cloudflare`, `jsDelivr`, and `Microsoft Ajax`), we differentiate between a cache HIT (solid box plots), and a cache MISS (hatched box plots), which are identified using two HTTP response header fields: `x-cache` and `cf-cache-status` (for `Cloudflare` [15]). The lack of hatched box plot for `Microsoft Ajax` indicates absence of cache misses, while for the remainder CDNs no distinction was possible.

If we focus on cache HIT, Figure 8(a) shows that the fastest download times are achieved via `jsDelivr`, which saves a minimum of 100 ms (when compared with `Cloudflare`) and up to 700 ms (when compared with `Highwinds`). Our results confirm that `jsDelivr` does indeed perform the best as they claim – since it relies on a network of CDNs. If we focus on cache MISS, the download time increases significantly, about 3x for both `Cloudflare` and `jsDelivr`. Still, even in presence of misses both CDNs manage to be faster than `Highwinds`.

Next, we focus on cache HIT only and we analyze the CDN performance by mobile network. Given we identified clear patterns by continent, Figure 8(b) shows one box plot for the total download time per continent: Europe, Australia, Asia, Central and South America, and Africa. The figure shows that, regardless of the CDN, the median download time grows by 10x (from 200-300ms up to 1-2 seconds) when comparing Europe and Africa. This result is the realization of networking effects we studied in the previous sections, such as high latencies and losses. For example the median latency (`mtr` to popular services as in Figure 6) for Ethio Telecom (Ethiopia), MTN Connect (Nigeria), and Telkom (Kenya) is 147ms, 108ms, and 149ms, versus 15ms, 23ms, and 50ms, in Play (Poland), Vodafone.de (Germany), and Lycamobile (France).

When comparing different CDNs, we confirm the high performance achieved by `jsDelivr`, which is only outperformed by `Google` and `Microsoft Ajax` in Africa. We can also see that `Highwinds` is competitive in Europe, Central and South America, and Africa, but suffers from very bad performance in Australia and Asia which skew its results when aggregating all locations (see Figure 8(a)).

Finally, Figure 9 shows how the MISSes are distributed across both `Cloudflare` and `jsDelivr` for the mobile networks that experienced cache MISSes. The results shows that for `Cloudflare` there is a very small probability of having a cache MISS. In contrast, for `jsDelivr` it can be observed that a number of mobile networks (Ethio Telecom in Ethiopia, Telkom in Kenya, Vodafone in Portugal, and SK Telecom in Korea) have almost a 100% cache MISSes for all tests performed at these locations.

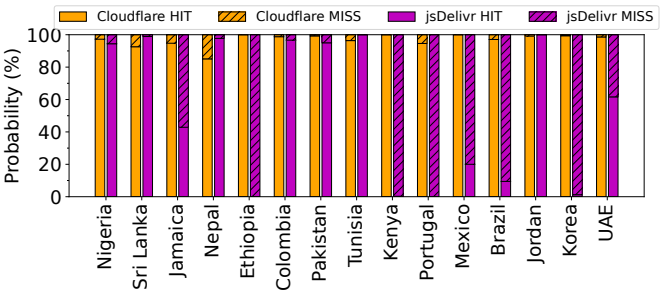


Fig. 9. Probability of cache HITs versus MISSes across mobile networks for `Cloudflare` and `jsDelivr` CDNs.

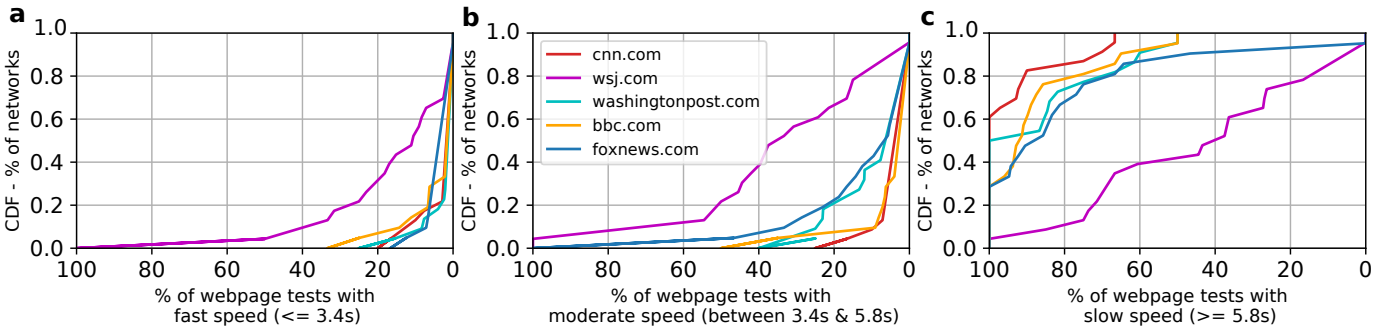


Fig. 10. CDFs of percentage of mobile networks with web browsing speed index, across five popular news websites, classified as: (a) fast ($\leq 3.4s$), (b) moderate (between $3.4s$ and $5.8s$), (c) slow ($\geq 5.8s$).

V. Application Performance Analysis

Web Browsing: Figure 10 shows Chrome UX style-based results for speed index [9] split by tested news website (see Section III-A for more information). We use Google's lighthouse definition [47] of page speed: fast ($<= 3.4s$), moderate (between $3.4s$ and $5.8s$), slow ($>= 5.8s$).

Figure 10(a) shows that *fast* webpage loads are rare: only 10% of the mobile networks have 10% of webpage loads faster than 3.4 seconds for four out of the five visited news web sites. The only exception is `wsj.com` which loads fast for 20% of the measurements from 40% of the mobile networks. Figure 10(b) also shows a slight increase in the number of networks with a significant number of *moderate* page loads: 20% of the networks have nearly 55%, 25% and 20% of their web tests considered to be of moderate speed for `wsj.com`, `foxnews.com`, and `washingtonpost.com`, respectively. Conversely, `bbc.com` and `cnn.com` still only have 10% of fast page loads in 10% of the mobile networks. Finally, 80%-100% of the webpages' tests are *slow* for 70% of the mobile networks (Figure 10(c))— except for `wsj.com`, again.

The reason of such slow loading times is that news websites are quite *heavy* for a low-end mobile device, *i.e.,* high CPU and memory usage. Not all news websites are the same, with `wsj.com` largely outperforming the rest, and `cnn.com` trailing, *i.e.,* it loads slow 90% of the times for 90% of the mobile networks. This is because `cnn.com` is one of the most complex pages with many embedded elements and recursive JavaScript. Given the importance of a diversified news outlet, such performance gap might have an impact on the user which goes far beyond their quality of experience.

Video Streaming: We now focus on video streaming via YouTube. Overall, a *good* streaming quality (480p–720p) is measured across most mobile networks. Lower quality levels are rare, with the exception of Lycamobile (France) and Claro (Colombia) where most streams only achieve 240p and 360p. We did not record any video stream at maximum resolution (1080p or 5 Mbps), despite many mobile operators offer download speeds which should support such quality. If we focus on the buffer playtime for mobile networks were high video qualities are detected, we observe median buffering time comprised between 30 and 60 seconds, which should be plenty to motivate the player to switch to a higher quality. We conjecture that the limited device resources have an impact on such switch, or some mobile operators adopt rate limits – which is not unlikely, as discussed in [34].

We next generalize the above observations by analyzing the percentage of times that a stream quality is detected at each mobile operator (see Figure 11). The figure shows that 1080p is indeed never detected, as discussed above, and that a *poor* quality (144p) is rare: only 20% of the mobile operators have up to 20% of streams at this quality. Indeed, the probability of higher quality streams tend to increase with the quality, e.g., half of the operators have at least 40% of the streams at 720p.

The above result suggests that YouTube performs much better across the different mobile networks in comparison to other previous conducted tests such web browsing. Given that YouTube quality does not highly depend on the network latency but rather on the available bandwidth (unlike web browsing). Ironically, we live in a world where it is much easier to watch a dancing cat[1] rather than browsing a news article online.

VI. Related Work

To the best of our knowledge, the closest related work to our study is [25] by Hung et. al. This work studies which factors impact user perceived performance of smartphones' network applications (Web browsing, video streaming, and voice over IP). The study was conducted in 2010 over the 3G networks of four major U.S. mobile operators: AT&T, Sprint, Verizon, and T-Mobile. In contrast, our paper studies and compares the performance of 24 operators across the globe. Further, we focus on 4G and introduce a broader set of networking experiments. Last but not least, we also propose a novel test-bed design based on regular Android devices.

With respect to test-bed design and deployment, MONROE [39] is the closest approach to `AmiGo`. MONROE is

[1] https://www.youtube.com/watch?v=NUYvbT6vTPs

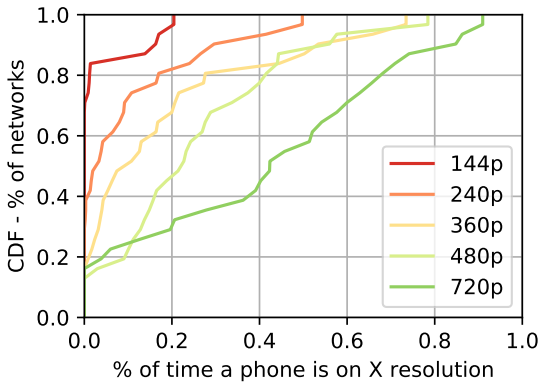


Fig. 11. Percentage of mobile networks with a YouTube tests' probability distribution of being on resolution comprised between 144p and 720p.

a measurement platform whose goal is to provide open-access assessment of the performance and reliability of mobile operators. MONROE is currently deployed across eleven mobile networks in four European countries. The core measurement component is called a *node* which is equipped with a Debian-based single-board computer plus three LTE modems connected to different providers. A centralized scheduling system allows MONROE users to post custom-made experiments to distributed nodes and remotely collect measurement results. In addition, each node independently executes certain background experiments, such as RTT measurements, to MONROE servers. Other large-scale research measurement platforms such as RIPE Atlas [41], BISmark [42], PlanetLab [14], and GENI [10] share common objectives with MONROE but are not designed for mobile environments.

Finally, crowd-sourcing approaches like Netalyzr [31], Net-Piculet [46], OpenSignal [2], RootMetrics [3] or MobiPerf [1] use custom-designed apps for measuring mobile operators via crowdsourcing. `AmiGo` uses a different approach since it allows full control of the device, e.g., even execution of classic Linux applications via `termux`, with no user interference. Note that while the only role of `AmiGo`'s volunteers is to carry the device, provide connectivity opportunities, and keep it charged, they occasionally interferred with our experiments as discussed in the next section.

VII. Conclusion and Lessons Learned

The lack of public data or test-bed makes it difficult to compare the user experience from mobile operators around the world. This paper has proposed a solution to this problem by introducing a new test-bed design called `AmiGo`. The design relies on travelers carrying mobile phones to act as measurement endpoints (MEs) and run a set of desired experiments. This design makes `AmiGo` easy to deploy, while offering realistic user mobility and devices. All code behind `AmiGo` [35]has been open sourced to help measurement researchers deploying their own test-beds in the wild.

To demonstrate the `AmiGo` capabilities, we have deployed it across five continents leveraging 31 students, and investigated the performance of 24 mobile networks. Among the many observations, we find that being *consistently* fast is challenging for a mobile operator, both in term of download speed and access latency. It follows that applications requiring high bandwidth and low latency may suffer from frequent slowdowns.

`AmiGo` was also employed in [44] to examine the effectiveness of various videoconferencing applications across WiFi and mobile networks worldwide. The study discovered that mobile clients demonstrate comparable patterns to fixed-access clients concerning round-trip time and video rate. Nevertheless, the overall quality of experience (QoE) at the application level is significantly affected by the limited processing power and screen size of mobile devices. `AmiGo` was also used in [26] to assess the performance of the two most popular location tags (Apple's AirTag and Samsung's SmartTag) through in-the-wild experiments conducted by having these tags as part of the vantage points carried by the amigos.

During the experiments conducted in this paper, we encountered a number of issues/challenges that we summarize below.

Battery Charging is Crucial: Testing mobile networks can place a considerable strain on battery life, particularly for low-end devices such as the `Redmi-Go`. As participants carried MEs without frequently interacting with them, it was common for the devices to run out of power unnoticed. To address this issue, we devised a strategy that would halt experiments when the battery level dropped below 15%, subsequently notifying participants through the installed app (see Figure 1(b)). This solution led to a 50% reduction in the amount of time measurement endpoints were unreachable due to dead batteries.

Debugging in the Wild: Given the highly dynamic nature of the `AmiGo` test-bed, we frequently encountered various unexpected issues. For example, ISP-specific policies or unanticipated participant behaviors (see below) that interfered with our measurements. To address these concerns, it was crucial to establish remote access to the MEs. Given all the devices in the field were situated behind ISP NATs, we leveraged a reverse SSH tunnel requested as an automation instruction (see Section II-A).

Unexpected Participants Behaviors: `AmiGo`'s MEs do not require user intervention, apart from setting up WiFi or mobile connectivity. However, we still had to deal with unexpected user interactions with the MEs. For example, while we ensured no sound is played by YouTube, some users opted to set their device in "no-disturb" mode which triggered a bug in our code when disabling audio. In turn, this caused some of our experiments to play sound causing distress (and complaints) in our volunteers. Similarly, some users turned off the screen of their device when they noticed an experiment was running, which could impact our results. We heavily rely on data processing and visual inspection of collected screenshots to ensure proper data sanitization. During the measurement campaign, we further sent reminders asking our volunteers to avoid interacting with the device, if possible.

REFERENCES

[1] Mobiperf. https://www.mobiperf.com/. Accessed: 2022-05-10.

[2] Open signal. https://www.opensignal.com/. Accessed: 2022-05-10.

[3] Rootmetrics. https://www.rootmetrics.com/. Accessed: 2022-05-10.

[4] Citi program - collaborative institutional training initiative. www.citiprogram.org, 2019. Accessed: 2022-05-18.

[5] Github. https://github.com/, Accessed: 2023-06-05.

[6] 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 *NSDI*, pages 367–380, 2015.

[7] APKMirror. Free and safe android apk downloads. https://www.apkmirror.com/.

[8] G. Arena. Xiaomi remi go. https://www.gsmarena.com/xiaomi_redmi_go-9534.php.

[9] C. Arsenault. Speed index explained - another way to measure web performance. https://www.keycdn.com/blog/speed-index.

[10] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[11] M. Chaqfeh, R. Coke, J. Hu, W. Hashmi, L. Subramanian, T. Rahwan, and Y. Zaki. Jsanalyzer: A web developer tool for simplifying mobile web pages through non-critical javascript elimination. *ACM Trans. Web*, 16(4), nov 2022.

[12] M. Chaqfeh, M. Haseeb, W. Hashmi, P. Inshuti, M. Ramesh, M. Varvello, L. Subramanian, F. Zaffar, and Y. Zaki. To block or not to block: Accelerating mobile web pages on-the-fly through javascript classification. In *Proceedings of the 2022 International Conference on Information and Communication Technologies and Development*, ICTD '22, New York, NY, USA, 2023. Association for Computing Machinery.

[13] M. Chaqfeh, Y. Zaki, J. Hu, and L. Subramanian. Jscleaner: De-cluttering mobile webpages through javascript cleanup. In *WWW*, 2020.

[14] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[15] Cloudflare. A global network built for the cloud. https://www.cloudflare.com/.

[16] D. Curry. Android statistics (2023). https://www.businessofapps.com/data/android-statistics/, 2023. Accessed: 2023-02-22.

[17] T. K. Dang, N. Mohan, L. Corneo, A. Zavodovski, J. Ott, and J. Kangasharju. Cloudy with a chance of short rtts: Analyzing cloud connectivity in the internet. In *IMC*, IMC '21, New York, NY, USA, 2021.

[18] A. Dev. Boot apps. com.argonremote.launchonboot.

[19] J. Dobbin. Lag! Top 5 Reasons your Ping is so High. https://www.hp.com/us-en/shop/tech-takes/5-reasons-your-ping-is-so-high, January 29, 2020.

[20] E. Enge. Mobile vs. Desktop Usage in 2020. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage.

[21] Fastly. Shielding . https://developer.fastly.com/learning/concepts/shielding/.

[22] M. Ghasemisharif, P. Snyder, A. Aucinas, and B. Livshits. Speedreader: Reader mode made fast and private. *CoRR*, abs/1811.03661, 2018.

[23] Google. Custom site performance reports with the crux dashboard. https://developers.google.com/web/updates/2018/08/chrome-ux-report-dashboard.

[24] B. Hesse. How to use youtube's 'stats for nerds'. https://lifehacker.com/how-to-use-youtubes-stats-for-nerds-1846125645.

[25] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Mobysis*, New York, NY, USA, 2010. Association for Computing Machinery.

[26] H. Ibrahim, R. Asim, M. Varvello, and Y. Zaki. I tag, you tag, everybody tags! *arXiv preprint arXiv:2303.06073*, 2023.

[27] jsDelivr. Open source cdn. how does it work? https://www.jsdelivr.com/network/infographic.

[28] B. Jun, M. Varvello, Y. Zaki, and F. E. Bustamante. Webtune: A distributed platform for web performance measurements. In *Proceedings of the 2021 Network Traffic Measurement and Analysis Conference (TMA)*.

[29] C. Kelton, M. Varvello, A. Aucinas, and B. Livshits. Browselite: A private data saving solution for the web. In *Proceedings of the Web Conference 2021*, WWW '21, page 305–316, New York, NY, USA, 2021. Association for Computing Machinery.

[30] M. Kimball. My traceroute (`mtr`). https://www.bitwizard.nl/mtr/.

[31] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proceedings of the 10th ACM conference on Internet measurement*, pages 246–259, 2010.

[32] J. Kupoluyi, M. Chaqfeh, M. Varvello, R. Coke, W. Hashmi, L. Subramanian, and Y. Zaki. Muzeel: Assessing the impact of javascript dead code elimination on mobile web performance. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 335–348, New York, NY, USA, 2022. Association for Computing Machinery.

[33] G. Labs. Graphana. https://grafana.com/.

[34] F. Li, A. A. Niaki, D. Choffnes, P. Gill, and A. Mislove. A large-scale analysis of deployed traffic differentiation practices. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 130–144. 2019.

[35] Y. Z. Matteo Varvello. Code for amigo tesbed. https://github.com/svarvel/mobile-testbed.

[36] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA, 2016. USENIX Association.

[37] OOKLA. SpeedTest Global Index. https://www.speedtest.net/global-index.

[38] OOKLA. Speedtest® CLI, Internet connection measurement for developers. https://www.speedtest.net/apps/cli.

[39] A. Özgü. Monroe: Measuring mobile broadband networks in europe. In *RAIM*, 2015.

[40] PostgreSQL. Postgresql: The world's most advanced open source relational database. https://www.postgresql.org/.

[41] R. N. Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3), 2015.

[42] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato. {BISmark}: A testbed for deploying measurements and applications in broadband access networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 383–394, 2014.

[43] termux. Android terminal emulator and Linux environment app. https://termux.com/.

[44] M. Varvello, H. Chang, and Y. Zaki. Performance characterization of videoconferencing in the wild. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 261–273, 2022.

[45] X. S. Wang, A. Krishnamurthy, and D. Wetherall. Speeding up web page loads with shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 109–122, Santa Clara, CA, 2016. USENIX Association.

[46] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. *ACM SIGCOMM Computer Communication Review*, 41(4):374–385, 2011.

[47] web.dev. Speed index. https://web.dev/speed-index/.

[48] Wikipedia. SARS-CoV-2 Omicron variant. https://en.wikipedia.org/wiki/SARS-CoV-2_Omicron_variant. Accessed: 2022/10/18.

[49] WPO-Foundation. Visual metrics. https://github.com/WPO-Foundation/visualmetrics.

[50] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu. From cloud to edge: A first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 37–53, New York, NY, USA, 2021. Association for Computing Machinery.