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

MATTEO VARVELLO, Nokia Bell Labs, USA
YASIR ZAKI, NYUAD, UAE

"Which performance can you expect at a 10$ cost from different mobile operators in the world?" This is a challenging question which is also the main motivation of our work. The question is challenging since no public data or test-bed is currently available which allows to answer it. Further, how do you define the performance of a mobile operator? Is it its download speed, probability to loose a packet, latency to popular providers...or all of the above? Our first contribution is the design and deployment of a test-bed with unprecedented footprint across mobile operators. The `AmiGo` test-bed relies on 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 deploy this test-bed through 31 travellers across 24 countries over a month (three countries had more than one traveller, but they all used the same operator within a country). Our second contribution is an analysis of the performance of 24 mobile operators *across the networking stack*, *i.e.,* from low level metrics like latency and download speed, to high level metrics like loading time of popular webpages. Our analysis shows that: a) 50% of the mobile networks have a 40-70% chance of encountering a low data rate, b) only 20% of mobile networks are characterized by low latencies (<20ms) to popular domains, c) networks across Asia, central/south America, and Africa have much higher CDN download times compared to Europe, exceeding a 10x increase in the case of Africa, d) most news websites load slowly, whereas YouTube achieves satisfactory performance, overall. Finally, our third contribution is the "Mobile Quality Index", a new metric which aggregates the above metrics into a single value, thus allowing a fair comparison among mobile operators.

1 INTRODUCTION

Mobile traffic has been on the rise in the last years, often surpassing its desktop counterpart. Data from Google Analytics' Benchmarking [1] shows that mobile devices drove 61% of visits to U.S. websites in 2020, up from 57% in 2019. Desktops were 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.

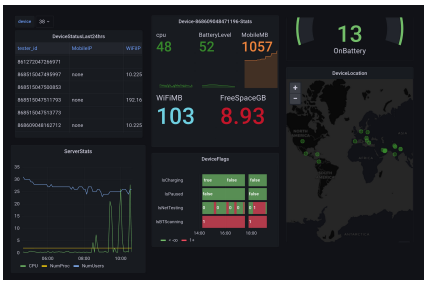
The research community has deeply investigated how to improve the performance of mobile users, e.g., by improving how fast pages load [12, 13, 20, 29, 32, 43], or reducing the data consumed [7, 26]. These works are motivated by the *challenging* conditions which mobile users face, such as low bandwidth or high network latencies. Few previous studies [24, 35] have measured performances across few mobile operators, mostly in the US or Europe, and for some specific application. To the best of our knowledge, no previous studies have yet explored a large number of operators across multiple continents while performing experiments *across the networking stack*, *i.e.,* from low level measurements like ping and speedtest up to application-layer measurements like video streaming and web browsing.

Performing such study is challenging due to the lack of a testbed which spans across many mobile operators. Even the popular MONROE [35] currently only spans 11 mobile networks in 4 European countries. Our first contribution is the design and development of the `AmiGo` testbed. The key idea behind this testbed is 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. The second contribution is a one-month measurement study of 31 mobile networks, which

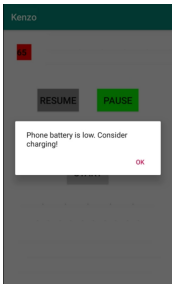
Authors' addresses: Matteo Varvello, Nokia Bell Labs, Murray Hill, NJ, USA; Yasir Zaki, NYUAD, Abu Dhabi, UAE.

2022. XXXX-XXXX/2022/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(a) Visual representation of the AmiGo testbed



(b) AmiGo's dashboard



(c) AmiGo Android app

Fig. 1. The AmiGo testbed: (a) worldwide distribution of measurement endpoint managed by the control server, (b) Graphana dashboard for real time monitoring, (c) mobile app to interact with volunteers carrying a measurement endpoint.

we use to propose the Mobile Quality Index metric (MQI) which summarizes and quantifies the performance across the mobile stack for multiple mobile networks around the globe.

The AmiGo testbed: It consists of *measurement endpoints*, low-end Android mobile phones (Redmi Go [9]), and a *control server*. These Android devices are rooted and equipped with `termux` [40] – an Android terminal emulator and Linux environment – thus allowing fine-grained instrumentation and data collection. The phones frequently report to the control server (running on AWS) their current status, e.g., connectivity (WiFi or mobile) and battery level. The controller is responsible to monitor the health of the phones, schedule experiments, and collect data.

Full stack networking experiments: We have deployed, via termux, a full stack performance tool consisting of speed-tests, traceroutes, HTTP GET of popular CDN objects, webpage loads, and YouTube tests. We further developed a mobile app which allows to interact with the "amigos", e.g., to run an experiment and provide feedback in real time.

The Mobile Quality Index metric (MQI): The above metrics are aggregated into a single value that allows a fair comparison among mobile operators. We call this aggregated number the "Mobile Quality Index" (MQI). It represents the mean quality of the mobile network performance by averaging different normalized metrics: 1) downlink and uplink rate, 2) latency, 3) DNS duration, 4) download time from CDNs, 5) speed index of popular web pages, and 6) YouTube's frame rate. Figure 14 shows the MQI performance comparison across the investigated mobile networks represented by their locations. The color of the circles represents the MQI score (from 0 to 1), and the size of the circles represents the cost of 1GB of data in US dollars for these mobile networks.

We recruited 31 university students to participate to the AmiGo testbed 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:

- Only a handful of mobile networks have fast download speeds exceeding 30 Mbps; specifically 20% of mobile networks show a 40% chance of providing such high speed.
- Apart from few African operators, DNS resolution time across the globe is mostly <100ms. Mobile performance when downloading content from CDNs varies significantly depending on location, for example performance in Africa were 10x slower than in Europe.

- The web browsing experience on the majority of mobile networks is slow. Four out of five tested news websites have 100% of the browsing performance marked as slow on 50% of the networks. In contrast, YouTube shows better performance across the board.
- In some countries (e.g., Telcel in Mexico and Flow in Jamaica) 1 GB of data costs about $4 and end-user performance is questionable. European countries have an overall high data cost but much higher performance, with the exception of Play (Poland) which gets the best of the two worlds: high performance at low cost (about $2 per GB).
- Performing experiments in the wild is hard! From dying batteries to ISP-specific behaviors, our test-bed (and code) was constantly under test. Remote debugging was probably the most important feature of the `AmiGo` test-bed to enable a successful measurement campaign.

2 TEST-BED ARCHITECTURE

This section presents the design and implementation of the `AmiGo` test-bed (see Figure 1a). The testbed consists of a *control server* that remotely manages a number of mobile *measurement endpoints* deployed worldwide. Each endpoint is an Android mobile phone (RedmiGo) which has been rooted and equipped with `termux` [40]. In the remainder of this section, we detail the functioning of both control server and measurement endpoints.

2.1 Control Server

The control server has three main tasks. First, it monitors the status (battery level, location, etc.) of the measurement endpoints. Second, it instruments the endpoints 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.

The control server is implemented in Python and runs on a powerful cloud server provided by InMotion hosting, with 16GB RAM and 6 CPUs. Such Python code enables restful APIs which the measurement endpoints call to: 1) report their current status (see Table 1 for a detailed description of the information reported), and 2) retrieve eventual instrumentation code. This code is also responsible to maintain a `postgres` [37] database which stores both device status updates and automation instructions. `Graphana` [30] is used to build a visual dashboard allowing an operator to identify potential issues with test-bed and/or experiments. Figure 1b shows an example of the dashboard status during our measurement study.

2.2 Measurement Endpoint

Rationale and Overview: A measurement endpoint is a mobile device that volunteers carry in their pocket while travelling. We chose a `Redmi-Go` [9] for three main reasons. First, it is an Android device, and Android is our mobile OS of choice since it allows fine-grained instrumentation. Second, it is a fairly cheap device (average retail price of $70), which is paramount since we are looking at purchasing and deploying a large number of devices. Third, despite being a low-end device (1.4 GHz quad-core CPU and 1 GB of RAM) our benchmarking shows little to no impact to most of our experiments (see Figure 3).

Android allows easy instrumentation via the Android Debug Bridge (ADB). That is, by connecting (via USB) a computing device (e.g., a small Raspberry Pi zero [36]), one can write code to automate operations (e.g., launch an app) and collect data (e.g., monitor CPU usage). This approach has, however, a few drawbacks. First, the computing device (and its battery) would increase the size of the measurement endpoint. Second, we are limited to the operations allowed via ADB. To solve the above limitations, we instead resort to rooting the Redmi Go and installing `termux` [40], an Android

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

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 (e.g., traffic collection). Further, the measurement endpoint becomes a self-contained device which is easy to carry.

Design and Implementation: The *core* code of a measurement endpoint has three key tasks. First, monitor the endpoint resources, e.g., battery level and connectivity. Second, decide when to run a list of pre-installed experiments (see Section 3.1). Third, interact with the control server, either to report its state or 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 described by Table 1. This information is then reported to the control server every 5 minutes.

With respect to *when* to run some pre-installed experiments, we adopt the following logic. Since our goal is to measure mobile networks, we run experiments when the user is only connected to a mobile network. We asked volunteers to connect the device to their home WiFi to avoid running most experiments from their home, where we expect them to spend the majority of the time (as confirmed by Figure 5b). On mobile, we schedule experiments every 30 minutes. We further monitor data consumption to avoid consuming more than 4 GB per day (see Section 3.2).

We leverage `cron` to ensure our code runs at each phone reboot, e.g., due to the device running out of battery. Further, we use it to force a device reboot each night at 2am. This allows to "clean" potential wrong states reached during the automation. We use the `Boot Apps` Android app [17] to ensure termux is launched at reboot, which in turns enable `cron` and `sshd`.

Fig. 2. Automated preparation and installation of the AmiGo's measurement endpoints.

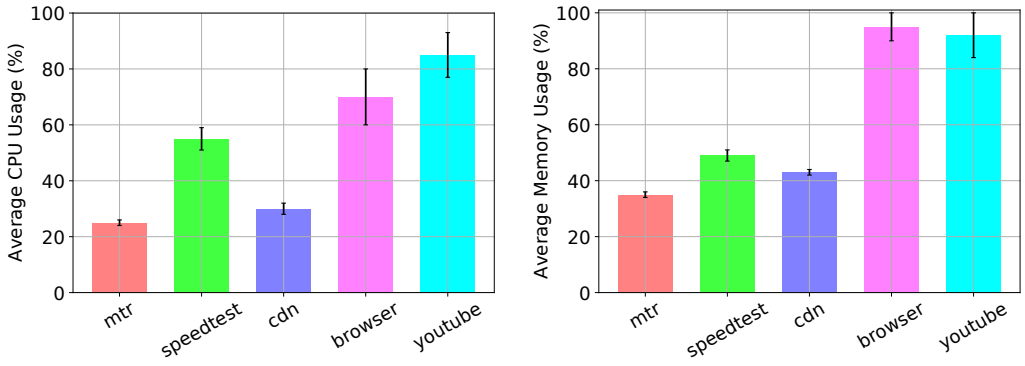
Code updates are handled via github [21]. Each device is instrumented with a private SSH key which has a double usage: pull code updates from github and open a reverse tunnel to the controller, if requested to do so. The latter is needed to allow access to a potentially misbehaving phone, or for debugging in the wild. In order to guarantee a consistent state across all devices, and to ease our job while preparing the devices, every app and termux package required was installed via script, and using local packages provided by APKMirror [8]. Figure 2 shows the installation script in action, installing and prepping the measurement endpoints.

Mobile App: In addition to the previous code, we have also developed a an application which is installed on each Redmi-Go. This application acts as the GUI of our experiments (see Figure 1c) and accomplishes three tasks. First, it shows the device identifier for simple communication with the developers in presence of a concern. Second, it allows a volunteer to pause an ongoing experiment. This is required when the user 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 allows to notify the user (via a notification on screen, see Figure 1c) that the mobile phone requires charging.

Google Account: Android devices require a Google account to function. 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 some random requests to verify our account, by entering its username and password. Fortunately, this operation can be automated, if detected. Via experimentation, we ended up building some automation which leverages YouTube to force a potential verification request, which is then detected by the presence of the MinuteMaidActivity instead of YouTube.

2.3 Limitations

The main limitation of the AmiGo test-bed is that it only supports Android. Extending to iOS (iPhone) is interesting, of course, but challenging due to few reasons. Technically speaking, automating third party iOS apps is currently not supported. Similarly, resource usage data like CPU and memory are not easily attainable in iOS. Clearly, this would limit our experiments and the amount of visibility we would gain. Logistically speaking, iPhones would significantly increase the cost of our test-bed.



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

Fig. 3. Benchmarking of CPU and memory usage at the measurement endpoint (`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 refer to the standard deviation.

A second limitation is that the `AmiGo` test-bed currently only consists of one specific Android phone: Redmi Go. We purposely focused on one device, to be able to compare results across locations. Still, it would be interesting to extend the results to more recent and powerful devices, e.g., to investigate 5G connections. This is an avenue for future work; further, we have open sourced our code which would allow other research groups to investigate such additional setups and research questions.

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 3a) and memory (Figure 3b) usage when performing the measurements described in Section 3.1. 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.

3 DATA COLLECTION

This section describes the experiments we devised for the `AmiGo` testbed which span the whole networking stack, from measuring network latencies to popular destinations, to web page load times. We first describe the experiments, and then provide an overview of the data collected.

3.1 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* [34], a Linux-native Speedtest tool backed by Ookla®. Speedtest CLI also reports latency, which we further augment via `mtr` using the technique discussed in the next paragraph. Finally, we comment on network losses capturing pcap files via `tcpdump` while loading popular webpages, *i.e.,* HTTP(S) and thus TCP traffic.

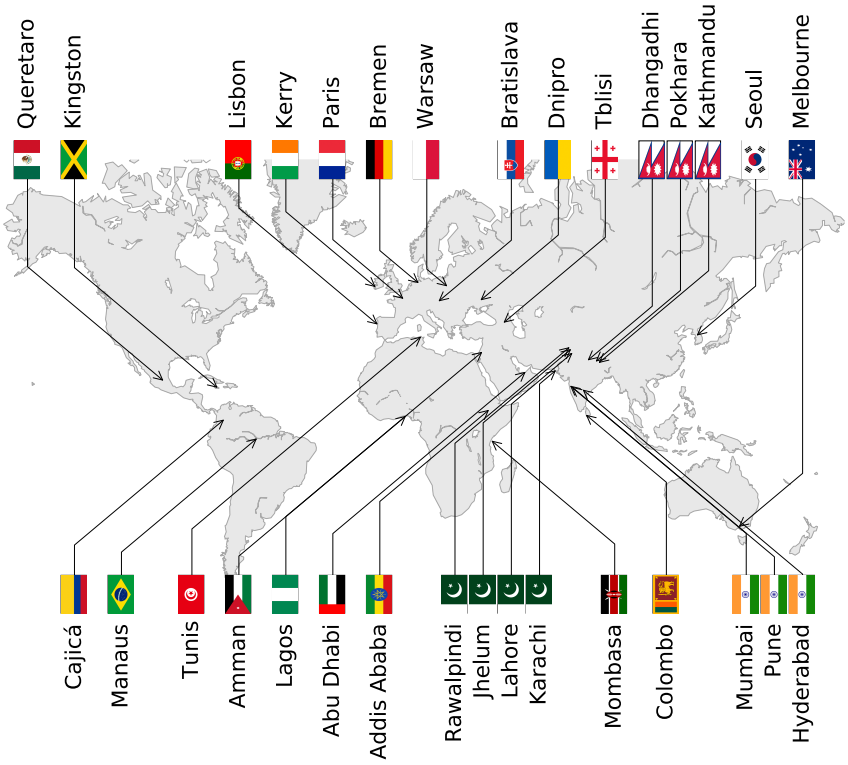


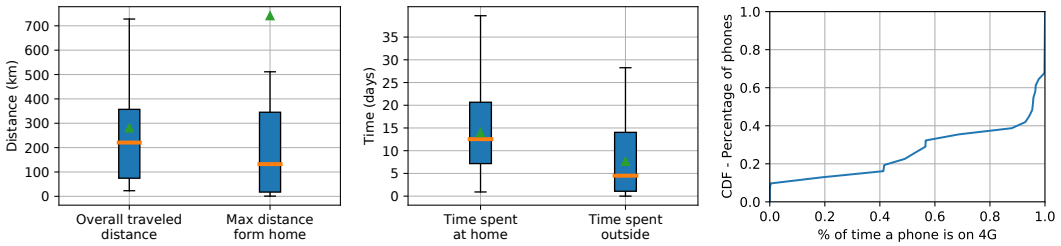
Fig. 4. Geographical distribution of AmiGo deployment between December 2021 and January 2022.

We use `mtr` [27] – 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 [16, 48], 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 measurement endpoints to use the DNS provided by the mobile network they are connected to. We use pcap files captured via `tcpdump` while loading popular webpages to extract DNS requests and analyze their duration.

CDN Performance: A Content Delivery Network (CDN) is a popular networking tool used to accelerate content retrieval, and thus 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 relying on a network of CDNs, where each request is processed by optimal CDN based on uptime and performance [25].

We iterate through these CDNs fetching `jquery.min.js` using `cURL` instrumented to report detailed statistics, such as duration of DNS resolution, TLS handshake, and total download time. We further collect HTTP headers report since, for some CDNs, they report whether the file was found in a cache, or not. Specifically, `Cloudflare`, `jsDelivr`, and `Microsoft Ajax` report cache *HIT*s or *MISS*es using two HTTP response header fields: `x-cache` and `cf-cache-status` (from



(a) Distance (km) (b) Time (days) (c) Probability for a phone to be connected to 4G.

Fig. 5. Analysis of user mobility and mobile access network type: (a) distance traveled and maximum distance from home box plots, (b) time spent at home vs. outside box plots, (c) probability of a measurement endpoint being on a 4G (LTE) mobile access, as a CDF of all measurements endpoints.

`Cloudflare` [15]). Given that `jsDelivr` relies on a network of CDNs including `fastly`, it reports additional information on cache *HIT*s or *MISS*es given that it uses the concept of "Shielding". When `fastly` sends the `x-cache` header, it sends two different statuses next to each other (e.g., '*HIT*, *MISS*', '*MISS*, *HIT*' etc.), where the second reports the status of the cache at the edge, and the first reports the status of the cache at the `shield`. A `shield` is a designated point of presence (POP) that collects requests from across the `fastly` network. In the presence of the `shield`, requests to the origin server will come only from the designated shield POP, and all other `fastly` locations will forward requests to the shield [19].

Application Performance: Web and video are two popular applications used on mobile devices. Most importantly, they 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 a video which is then fed to visualmetrics [47] to extract performance timing metrics such as the popular SpeedIndex [46], which shows how quickly the contents of a page are visibly populated. We further collect `pcap` traces while performing Web experiments.

For video, we automated the YouTube app. YouTube allows to enable "stats-for-nerds" [23] 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. Unfortunately, enabling such mode is tricky as it requires a set of "clicks", but if the state of the app is not ready, e.g., due to extra load on the phone or slow network, such clicks can be incorrect. While we cannot infer the state of the app, since one single activity is used, we at least can verify whether useful information is dumped to a file, and thus either re-attempt enabling the stats or interrupt the experiment. 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).

3.2 Data Overview

World Coverage: We have deployed `AmiGo` measurement endpoints via 31 university students while travelling back home during the 2021 winter break. The students were recruited based on the home countries that they were traveling back to, as well as their willingness to carry the phones with them at all times – or as much as possible – without using or interacting with the phones. The data collected spans 31 cities in 24 countries (see Figure 4) between December 15th, 2021 and

January 30th, 2022. In three of the countries we had more than one student, each traveling to a different city, namely: Pakistan (4 students), India (3 students), and Nepal (3 students). The students were instructed to purchase a mobile SIM card from their own home country 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.

Mobile Access Type: Figure 5c shows the probability of phones that encountered a 4G mobile access network (LTE) during the measurement campaigns, as well as, the probability of how many of the phones' tests were done on 4G. The figure shows that 60% of the phones had 90% of their measurements tests performed on a 4G mobile access network (i.e., the 0.4 - 1.0 portion of the CDF in the y-axis of the figure). 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. Figure 5a summarizes the users mobility with respect to their home, *i.e.,* the location where they spent the majority of the time (12.5 days, on the median). This analysis is useful since time spent outside of the house provides us with an opportunity to perform mobile measurements in the wild.

The figure shows that most users were quite mobile, exploring more than 130 km over their data collection period, different per user as shown in Table 2 (in the appendix), and spending about 100 hrs outdoor (roughly 4.5 days). Note that some users ventured in trips, which explains the upper whisker in Figures 5a and 5b. For example, one user located in France went to a trip to Spain (see Figure 17(a) in the appendix) which explains such large "distance from home". Similarly, few users spent very little time outside, mostly due to safety restrictions caused by the rapid spread of Omicron (COVID-19 variant) [5].

Data Usage: Table 2 shows, among other things, the total mobile data consumption per user. Data consumption depends on two factors: 1) how much users move (see Figures 5a and 5b), 2) the 4 GB limit per day we set (see Section 2.2). 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. Such high usage was driven by a combination of high user activity (see Figure 17(c)) and increased trip duration due to COVID-related travel restrictions.

3.3 Ethics

Given that we recruited participants to carry custom prepped mobile devices around, 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 [6] certified. Participants were also provided with a consent form to read, and sign, acknowledging their willingness to participate. They were given the opportunity to ask questions about the study and what was being collected.

We asked the participants to mainly carry these mobile phones with them, charge them, install a mobile data sim-card, as well as, 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 did inform the participants before hand about this concern and we obtained their written consent that they approve this collection. As such, we believe that this are deemed to be of low-risk.

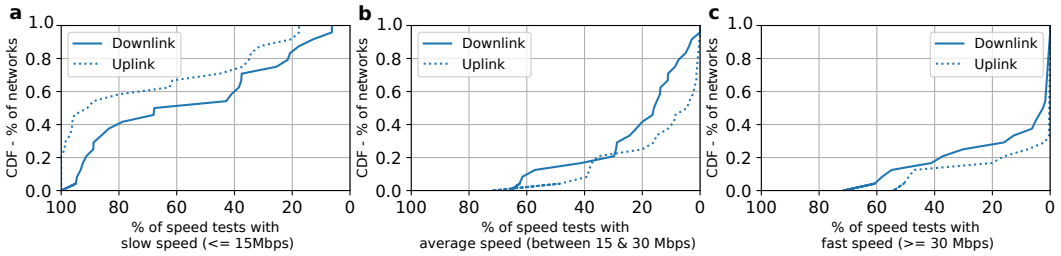


Fig. 6. Speed tests' measurements results showing the CDF of percentage of mobile networks with downlink and uplink speeds quantified as: (a) slow (<=15Mbps), (b) average (between 15 and 30 Mbps), (c) fast (>=30Mbps).

4 NETWORK CHARACTERIZATIONS

This section focuses on the analysis of low-level networking experiments: speedtests, traceroutes, DNS lookups, TLS performance, loss probability, and simple HTTP GET from popular CDNs. Figure 15 summarizes the results of each experiment per mobile operator – minus CDN results for which the analysis is more complex and thus split to its own Figure 10. Each metric is presented in the form of a box plot containing data from the many experiments performed in a given mobile operator. Please refer to Table 2 for a summary of the number of samples per experiment, and mobile operator.

Figure 15 has to be seen as a visualization of the data-set we have collected, and that we are open sourcing. It has value to answer high level questions like: *"does Vodafone offer similar download speeds across countries?"* The answer is yes for Europe – median download speed of 40-45 Mbps measured in Germany, Ireland, and Portugal – but no for Australia where only 15.5 Mpbs are measured. Commenting on this figure in general is hard, which is why we have devised the following methodology.

For a given metric, e.g., download speed, our data-set contains multiple data points for each mobile operator. It follows that one way to visualize the download speed is to show the median download speed, for example, across all operators. We use such visualization and further augment it with the cumulative distribution function (CDF) of the percentage of mobile networks for which a target metric is within a range, e.g., *slow* download speed as defined by less than 15 Mbps. This analysis, inspired by Chrome User Experience Report (CRuX) [22], further allows us to comment on the probability, expressed in number of tests, that a certain condition is met at a fraction of the mobile networks.

4.1 Speed Tests

Figure 6 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, a fast downlink/uplink has a speed equal or higher than 30 Mbps. We chose these thresholds using indications from SpeedTest Global Index which ranks 140 operators by their upload/download speeds [33]

As expected and also reported in [33], Figure 6 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

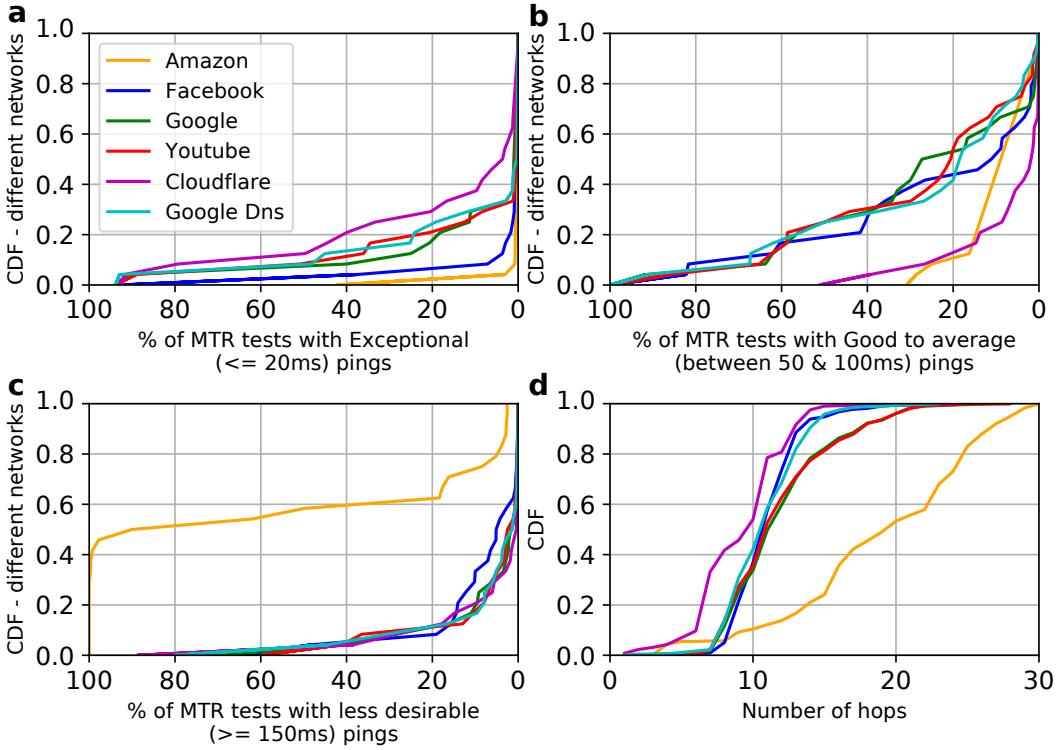


Fig. 7. MTR pings results showing the CDFs of percentage of mobile networks, per server, with latencies quantified as: (a) exceptional (<=20ms), (b) good to average (between 50ms and 100ms), (c) less desirable (>=150ms). Additionally, (d) showing the MTR trace-routes results as CDFs of the number of traveled hops till destination, split per tested server.

(e.g., Flow in Jamaica, see Figure 15) . 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., high quality video streams, might suffer from frequent slowdowns.

4.2 MTR Pings and Traceroute

Figure 15 shows box plots of the network latency (green), *i.e.,* ping component of `mtr`, experienced by different mobile networks when accessing popular DNS and content providers. For most networks, the median latency is below 100 ms, apart from few African operators (Ethio Telecom in Ethiopia, MTN Connect in Nigeria, and Telkom in Kenya) where the median latency exceeds 100 ms, and at times peaks above 200 ms. At the other end of the spectrum, many European operators (Play in Poland, Orange in Slovakia, and Vodafone both in Germany and Portugal) offer very low latencies in the range of 20 ms.

Next, Figure 7 (a), (b), and (c) show respectively the aggregated results based on the quality of the mobile network latency tests [18]: 1) *exceptional* ping which falls below 20 ms, 2) *good* to *average* ping between 50 and 100 ms, and 3) *poor* 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`.

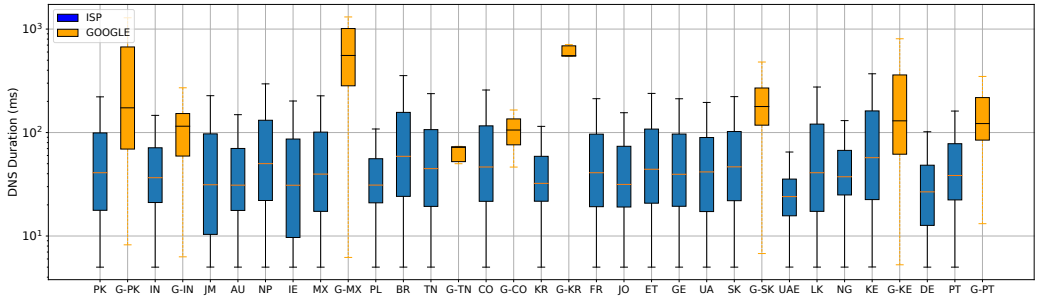


Fig. 8. DNS lookup times' box plots split per mobile operator, comparing when the phones used local DNS servers (powered by the mobile operator), vs. when it used Google DNS (i.e. 8.8.8.8 or 8.8.4.4).

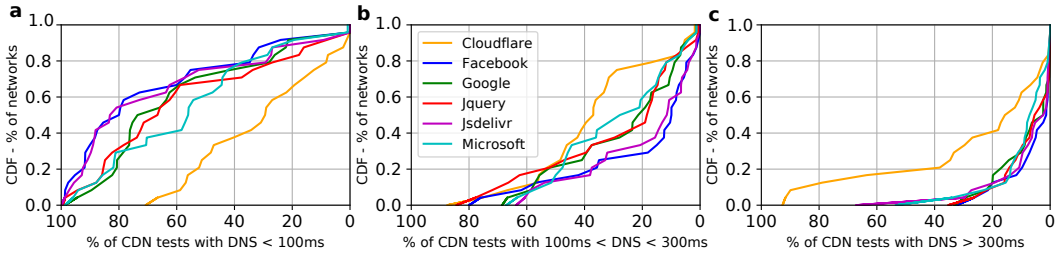


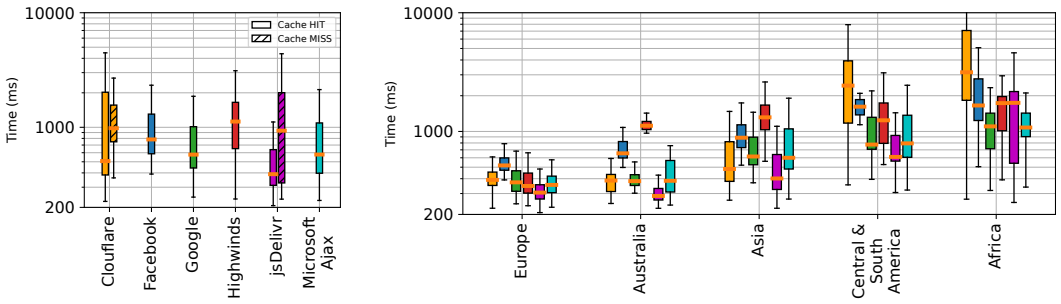
Fig. 9. DNS resolution time results based on the CDN measurements tests, showing the CDFs of percentage of mobile networks, per CDN, with timings quantified as: (a) fast (<=100ms), (b) moderate (between 100ms and 300ms), (c) slow (>=300ms).

Figure 7 (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 7 (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 poor (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 7 (d) shows the median number of hops for `Amazon` is nearly double that of the other servers. This implies that `Amazon` servers are located geographically further away for most mobile operators, which suggests a lack of `Amazon` CDNs at different locations across the globe.

4.3 Domain Name System

Figure 8 shows box plots of DNS lookup times—extracted from Web browsing tests (see Table 2) — per mobile operator. Blue box plots refer to lookup times obtained when, to the best of our



(a) Per cache HIT/MISS (b) Per continent

Fig. 10. Total download time box plots for six popular CDN servers: (a) overall comparison of the download time for each CDN, including the impact of cache *MISS*es (for `Cloudflare` and `jsDelivr`), (b) comparison of the download time split per five continent for each CDN server.

knowledge a *local* (powered by the mobile operator) DNS was 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 the mobile operators; it follows that several operators (e.g., Telenor in Pakistan, IND Airtel in India, and Telcel in Mexico) sometimes rely on Google DNS.

Overall, Figure 8 shows that Google DNS is not a good choice for the mobile operators which adopted it, as the median DNS lookup time doubles (or more, see a 10x increase for Telcel in Mexico). Conversely, the median DNS duration for operator-provided DNS is overall quite fast, about 40ms, suggesting that most DNS queries are cached and solved locally. This is likely due to the high popularity of the websites under test (see Section 3.1); we conjecture that Google DNS is used only for a small portion of the customers (and traffic) thus benefiting less from caching. When comparing mobile operators, their DNS achieve overall comparable performance.

Next, we focus on DNS lookup time as measured during CDN tests (Figure 9). Except for `Cloudflare`, the majority of the networks (and tests) are characterized by fast lookup times, thus matching the previous result in the context of Web browsing. For example, 50% of the mobile networks have 60-80% of the DNS lookup times for CDN testing < 100ms, if ignoring `Cloudflare`. Although not shown, the median lookup times for these experiments sits around 90ms, aka 2.25x more than above. This is not surprising given that CDN providers are known for forcing short TTL, thus reducing the chance for an operator-provided DNS to respond from its cache. The remainder of the experiments are mostly average, *i.e.,* between 100 and 300 ms, and only `Cloudflare` has some poor DNS lookup times of more than 30 ms, e.g., for 20% of the mobile networks with at least 40% of the experiments.

4.4 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 TLS version 1.3 using various cipher suites per CDN.

We start by comparing the performance of each CDN globally, *i.e.,* aggregating measurements from 31 mobile networks (see Figure 10a). 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 `Facebook`, `Google`, and `Highwinds` is due to a lack of information with respect to cache *HIT* and *MISS*. Conversely, for `Microsoft Ajax` it indicates that no cache misses were measured.

If we focus on cache *HIT*, Figure 10a 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`). As explained earlier, we chose `jsDelivr` due to its promise to provide better performance by relying on a network of CDNs. Our results confirm that `jsDelivr` does indeed perform the best as they claim. 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 10b shows one box plot for the total download time per continent: Europe[1], 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 concretization of networking effects we studied in the previous section, such as high latencies and losses. See for example latencies for Ethio Telecom in Ethiopia, MTN Connect in Nigeria, and Telkom in Kenya with median values of 147ms, 108ms, and 149ms, respectively (see Figure 15).

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 10a).

Finally, Figure 11 shows how the *MISS*es are distributed across both `Cloudflare` and `jsDelivr` for the mobile networks that experienced cache *MISS*es. 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 *MISS* for all tests performed at these locations.

We conclude with an interesting finding. The expected size of the file downloaded (`jquery.min.js`) is 89,501 KB. All CDNs—apart from `facebook`—served back the exact file size all the time. `Facebook` CDN served variable file sizes with a median size of roughly 87 KB, and as low as 78 KB, and as high as 97 KB. We don't know the reason behind this variation in the file sizes given that such JavaScript files are usually both minified and uglified to reduce their sizes, which makes comparing them very hard.

5 APPLICATION PERFORMANCE ANALYSIS

This section focuses on the analysis of application level experiments: Web browsing, and YouTube.

5.1 Web Browsing

Figure 12 shows the aggregated Chrome UX style-based results for speed index [10] split by tested news website (see Section 3.1 for more information). We use Google's lighthouse definition [46] of page speed: fast (<= 3.4s), moderate (between 3.4s and 5.8s), slow (>= 5.8s).

[1]We excluded Ireland's results from Europe because most tests (72%) were conducted over 3G which would end up biasing the results.

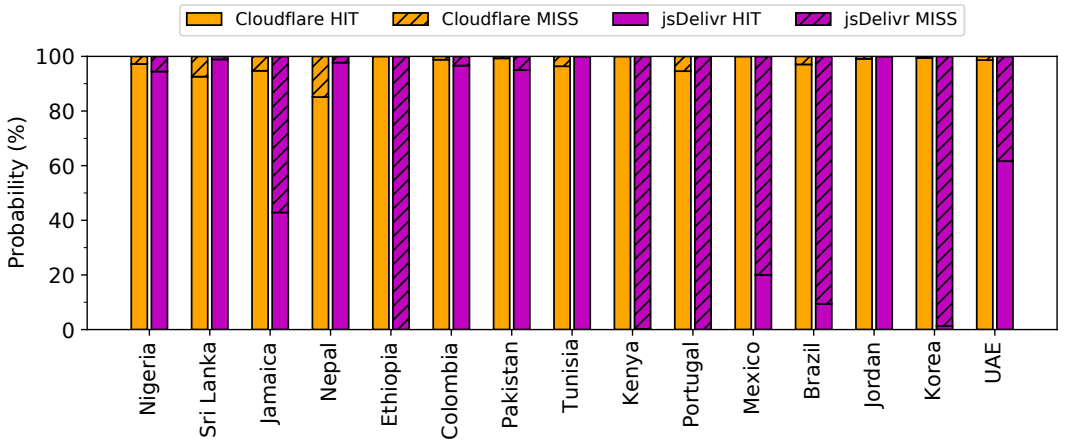


Fig. 11. Probability of cache *HIT*s vs. *MISS*es across mobile networks for `Cloudflare` and `jsDelivr` CDNs.

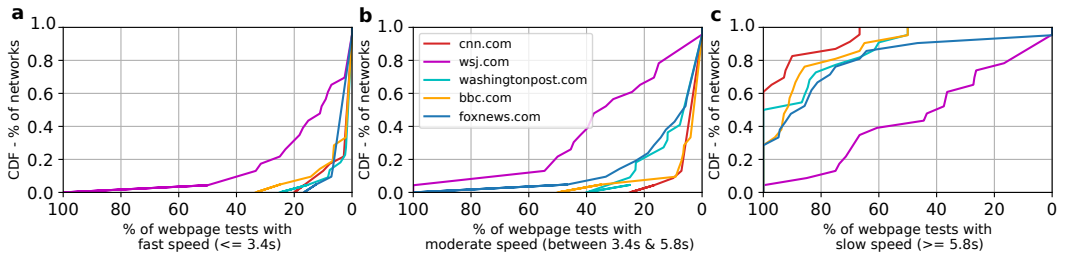


Fig. 12. Web browsing results represented by speed index CDFs (as a function of mobile networks); split by five popular news websites, with web performance quantified as: (a) fast (<=3.4s), (b) moderate (between 3.4s and 5.8s), (c) slow (>=5.8s).

Figure 12(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. Next, Figure 12(b) 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 12(c))—again with the exception of `wsj.com`.

In summary, news websites are mostly too heavy for a low-end mobile device – due to its limited resources (see Figure 3) – especially when located in a developing region. 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.

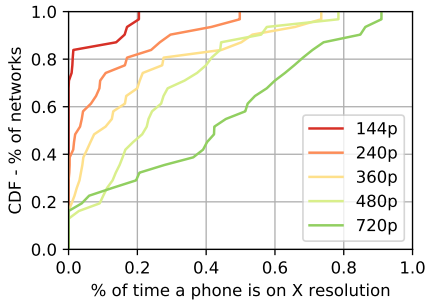


Fig. 13. Percentage of mobile networks with a YouTube tests' probability distribution of being on a X YouTube resolution. Showing a CDFs for each of the five different resolutions (144p to 720p).

5.2 YouTube

We now focus on another popular mobile application: video streaming via YouTube. Figure 16 shows a summary of this analysis consisting of: a) the probability of a YouTube stream to be on a given resolution, from very low (144p, depicted in red) to very high (1080p, depicted in dark green), and b) box plots of YouTube's buffered playtime. 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 (see Figure 15). If we focus on the buffer playtime for mobile networks were high video qualities are detected, the figure shows 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 [31].

We next generalize the above observations by analyzing the percentage of times that a stream quality is detected at each mobile operator (see Figure 13). 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 results 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). Funny enough, we live in a world where it is much easier to watch a dancing cat[2] rather than browsing a news article online.

6 MOBILE QUALITY INDEX

The previous two sections evaluated how mobile networks measured by `AmiGo` fared over a wide variety of tests and metrics. The evaluation spanned the whole networking stack, from low layer metrics (such as latency, throughput, and losses) to the speed of popular network elements like DNS and CDN, all the way to application performance in the form of web browsing and YouTube tests. One angle that we did not explore yet is the monetary component of a mobile connection. More

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

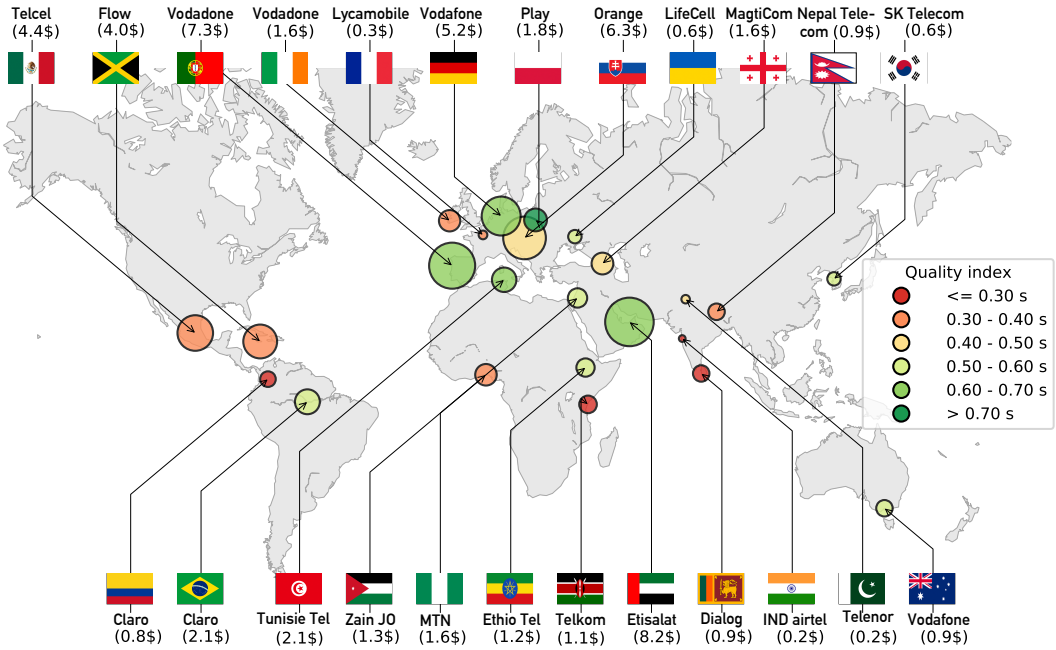


Fig. 14. Mobile Quality Index (MQI) of the different mobile operators. The size of the circles represents the cost in $, whereas the color represents the MQI value.

specifically, we want to answer the main question which motivated our paper (see the abstract): "*Which performance can you expect at a 10$ cost from different mobile operators in the world?*"

To answer this question, it is important to consolidate all results obtained so far into a single metric which can be easily compared across the different mobile networks we studied around the globe. For that, we propose the "Mobile Quality Index" (MQI), a single unit-less number that represents the normalized mean of all the metrics measured via our experiments. To compute the MQI, the mean of each metric per mobile network is first individually calculated, and then normalized with the maximum mean observed across all networks for that specific metric. The metrics used are: 1) downlink and uplink rate, 2) latency, 3) DNS duration, 4) download time from CDNs, 5) speed index of popular web pages, and 6) YouTube's frame rate.

For metrics where "more is better" (e.g., downlink and uplink rate, and YouTube's frame rate), the metrics are normalized by dividing the mean by the maximum observed mean across all networks, as:

$$\tilde{m_i} = \frac{m_i}{\max\limits_{j \in \text{all operators}} (m_j)} \tag{1}$$

For metrics where "less is better" (e.g., latency, DNS lookup time, speed index, and CDN download time), we normalize by taking the complement of dividing the metric's mean by the maximum observed mean across all networks, as:

$$\tilde{m_i} = 1 - \frac{m_i}{\max\limits_{j \in \text{all operators}} (m_j)} \tag{2}$$

In Equation 1 and 2, $\tilde{m}$ is the normalized metric, m is the targeted metric, i is the targeted mobile operator, j is all operators.

Finally, MQI is computed as the mean of all of the above seven normalized metrics. Figure 14 shows the the MQI against the cost of 1 GB of data obtained directly from AmiGo participants. Each circle in the figure carries two distinct information: a) the size represents the cost of 1 GB of data (the bigger the costlier), and b) the color is the MQI, representing the performance quality (the greener the better). The results shows that for a number of operators such as Telcel (Mexico) and Flow (Jamaica), the cost of 1 GB is high and the performance quality is low. For other operators – such as Claro (Colombia), Telkom (Kenya), Dialog (Sri Lanka), and IND Airtel (India) – the cost of data is low but the performance is far from ideal (dark red color circles). Many European operators – such as Vodafone (both in Germany and Portugal) and Orange in Slovakia – have high data costs but also overall good performance. Finally, Play (Poland) achieves the best performance: lowest cost for data and highest MQI.

7 RELATED WORK

To the best of our knowledge, the closest related work to our study is [24] by Hung et. al, which studied in 2010 which factors impact user perceived performance of smartphones' network applications. The study was conducted over the 3G networks of four major U.S. mobile operators: AT&T, Sprint, Verizon, and T-Mobile. In contrast, this paper studies and compares the performance of 24 operators across the globe. Further, this study focuses on 4G and introduces a broader set of networking experiments, e.g., involving CDN tests, Web browsing, and video streaming.

With respect to test-bed design and deployment, MONROE [35] is the closest approach to AmiGo. MONROE is a measurement platform that enables reliable open-access assessment of the performance and reliability of 11 mobile networks in 4 countries in Europe. The system's core design is a customized MONROE node with a Debian-based single-board computer plus three LTE modems connected to different providers. A centralized experiment 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 [38], BISmark [39], or PlanetLab [14] GENI [11], share multiple common objectives with MONROE, but they are not designed to operate in the mobile environment. However, several crowd-sourcing approaches like Netalyzr [28], NetPiculet [45], OpenSignal [3], RootMetrics [4] or MobiPerf [2], have emerged over the years leveraging the broad adoption of mobile devices globally and rely on the willingness of end-users to run the proposed tests. Utilizing these tools aims to identify and monitor significant metrics that can accurately define mobile networks performance. Several research projects use custom-designed apps for crowd-sourcing and measuring mobile operators performance and popular Internet applications, focusing mainly on web browsing [44] and video streaming [42].

AmiGo shares a number of commonalities with MONROE, especially the vision of relying on measurements endpoints carried by volunteers to asses and measure mobile networks performance. However, AmiGo has a number of advantages over MONROE, such as: a) AmiGo measurements endpoints have higher mobility, evident by the wide range of our participants' movements (see Figure 17), b) in contrast to MONROE's 4 locations, AmiGo extends the measurements to 24 countries (31 cities) spanning 5 continents, and c) AmiGo use a low-end Android phone, where the measurements results are representative of actual performance quality experienced by users.

8 DISCUSSION AND CONCLUSION

This paper presented AmiGo, a large scale testbed consisting of many measurements endpoints realized through rooted Android mobile phones with both cellular and WiFi connectivity. AmiGo is capable of measuring a large suite of performance measures ranging from low level metrics such as speedtests, latencies, packet losses etc., all the way to higher level performance of applications such as browsing the Web, or playing a YouTube video. AmiGo can be extended to offer many additional measurements, or even allowing other researchers to run their own custom experiments as already done in [41]. The AmiGo network consists of 31 Redmi-Go Android phones carried by students travelling back home during the 2021 winter break. The AmiGo architecture allows for deploying and installing new tests on-the-fly through the AmiGo control server and GitHub source code. We plan to open-source the AmiGo test-bed, and offer experiments to researchers to run their own measurements.

8.1 Lessons Learnt

During the experiments conducted for this paper between December 2021 and January 2022, we encountered a number of issues/challenges that we summarize below.

Battery Charging is Crucial: one of the biggest challenge we encountered was dying batteries. Given that the participants were only carrying the phones with them without regularly interacting with it, it was often the case that the phone ran out of charge without them noticing. Mobile networks testing does have its toll on battery life, especially for a low-end device like Redmi-Go. As a countermeasure, we implemented a logic that stops experiments when the battery level is below 15%, and notifies the participants through the installed app to charge their phone (see Figure 1c). This resulted in a 50% reduction of the time vantage points were unreachable due to a dead battery. In the future, we plan to increase the battery life of a measurement endpoints by providing an external power bank integrated with the device's case.

Debugging in the Wild: given the extreme dynamic nature of the AmiGo test-bed, and the fact that it spans 5 different continents with many countries, we often encountered many unexpected issues. Some where related to certain ISPs' policies that interfered with some of our measurements, and others were unexpected participants behaviors that caused certain elements to stop working (see below).

To deal with the above issues, it was important to have some remote access to the phones. The challenge was that all of these phones in the wild are behind ISP NATs, which of course hindered the ability to SSH to the phones directly and debug. Our solution was the establishment of a reverse SSH tunnel requested to the vantage point as an *automation instruction* (see Section 2.1). This allowed for an easier remote debugging sessions when encountering problems in the wild.

Unexpected participants behavior: Despite AmiGo's vantage points do not require user intervention, apart from setting up WiFi or mobile connectivity, we still had to deal with unexpected user behaviors. 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 triggered some of our experiments to play sound causing distress (and complaints) in our voulunteers. Similarly, some users turned off the device screen when they noticed an experiment was running, which could also impact our results. We heavily rely on data processing and visual inspection of collected screenshots to ensure proper data sanitization.

REFERENCES

[1] Mobile vs. Desktop Usage in 2020. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage.
[2] Mobiperf. https://www.mobiperf.com/.
[3] Open signal. https://www.opensignal.com/.
[4] Rootmetrics. https://www.rootmetrics.com/.
[5] SARS-CoV-2 Omicron variant. https://en.wikipedia.org/wiki/SARS-CoV-2_Omicron_variant. Online; accessed 18 October 2022.
[6] Citi program - collaborative institutional training initiative. www.citiprogram.org, 2019. Accessed: 2022-05-18.
[7] 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 , pages 367–380, 2015.
[8] APKMirror. Free and safe android apk downloads. https://www.apkmirror.com/.
[9] G. Arena. Xiaomi remi go. https://www.gsmarena.com/xiaomi_redmi_go-9534.php.
[10] C. Arsenault. Speed index explained - another way to measure web performance. https://www.keycdn.com/blog/speed-index.
[11] 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.
[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 , 2022.
[13] M. Chaqfeh, Y. Zaki, J. Hu, and L. Subramanian. Jscleaner: De-cluttering mobile webpages through javascript cleanup. In , pages 763–773, 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] 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 '21*, page 62–79, New York, NY, USA, 2021. Association for Computing Machinery.
[17] A. Dev. Boot apps. https://play.google.com/store/apps/details?id=com.argonremote.launchonboot&hl=en&gl=US.
[18] 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.
[19] fastly Developers. Shielding. https://developer.fastly.com/learning/concepts/shielding/#debugging.
[20] M. Ghasemisharif, P. Snyder, A. Aucinas, and B. Livshits. Speedreader: Reader mode made fast and private. *CoRR*, abs/1811.03661, 2018.
[21] github. Github. https://github.com/, 2022.
[22] Google. Custom site performance reports with the crux dashboard.
[23] B. Hesse. How to use youtube's 'stats for nerds'. https://lifehacker.com/how-to-use-youtubes-stats-for-nerds-1846125645.
[24] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *MobiSys '10*, page 165–178, New York, NY, USA, 2010. Association for Computing Machinery.
[25] jsDelivr. Open source cdn. how does it work? https://www.jsdelivr.com/network/infographic.
[26] C. Kelton, M. Varvello, A. Aucinas, and B. Livshits. Browselite: A private data saving solution for the web. *arXiv preprint arXiv:2102.07864*, 2021.
[27] M. Kimball. My traceroute (MTR). https://www.bitwizard.nl/mtr/.
[28] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In , pages 246–259, 2010.
[29] T. Kupoluyi, M. Chaqfeh, M. Varvello, W. Hashmi, L. Subramanian, and Y. Zaki. Muzeel: A dynamic javascript analyzer for dead code elimination in today's web. *arXiv preprint arXiv:2106.08948*, 2021.
[30] G. Labs. Graphana. https://grafana.com/.
[31] 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.
[32] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In , Santa Clara, CA, 2016. USENIX Association.
[33] OOKLA. SpeedTest Global Index. https://www.speedtest.net/global-index.
[34] OOKLA. Speedtest® CLI, Internet connection measurement for developers. https://www.speedtest.net/apps/cli.
[35] A. Özgü. Monroe: Measuring mobile broadband networks in europe. In , 2015.
[36] R. Pi. Raspberry pi zero w. https://www.raspberrypi.com/products/raspberry-pi-zero-w/.
[37] PostgreSQL. Postgresql: The world's most advanced open source relational database. https://www.postgresql.org/.
[38] R. N. Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3), 2015.

[39] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato. {BISmark}: A testbed for deploying measurements and applications in broadband access networks. In , pages 383–394, 2014.
[40] termux. Android terminal emulator and Linux environment app. https://termux.com/.
[41] M. Varvello, H. Chang, and Y. Zaki. Performance characterization of videoconferencing in the wild. In , 2022.
[42] F. Wamser, M. Seufert, P. Casas, R. Irmer, P. Tran-Gia, and R. Schatz. Yomoapp: A tool for analyzing qoe of youtube http adaptive streaming in mobile networks. In , pages 239–243. IEEE, 2015.
[43] X. S. Wang, A. Krishnamurthy, and D. Wetherall. Speeding up web page loads with shandian. In , pages 109–122, Santa Clara, CA, 2016. USENIX Association.
[44] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In , pages 31–40, 2012.
[45] 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.
[46] web.dev. Speed index. https://web.dev/speed-index/.
[47] WPO-Foundation. Visual metrics. https://github.com/WPO-Foundation/visualmetrics.
[48] 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 *IMC '21*, page 37–53, New York, NY, USA, 2021. Association for Computing Machinery.

A APPENDIX

A.1 Additional Results

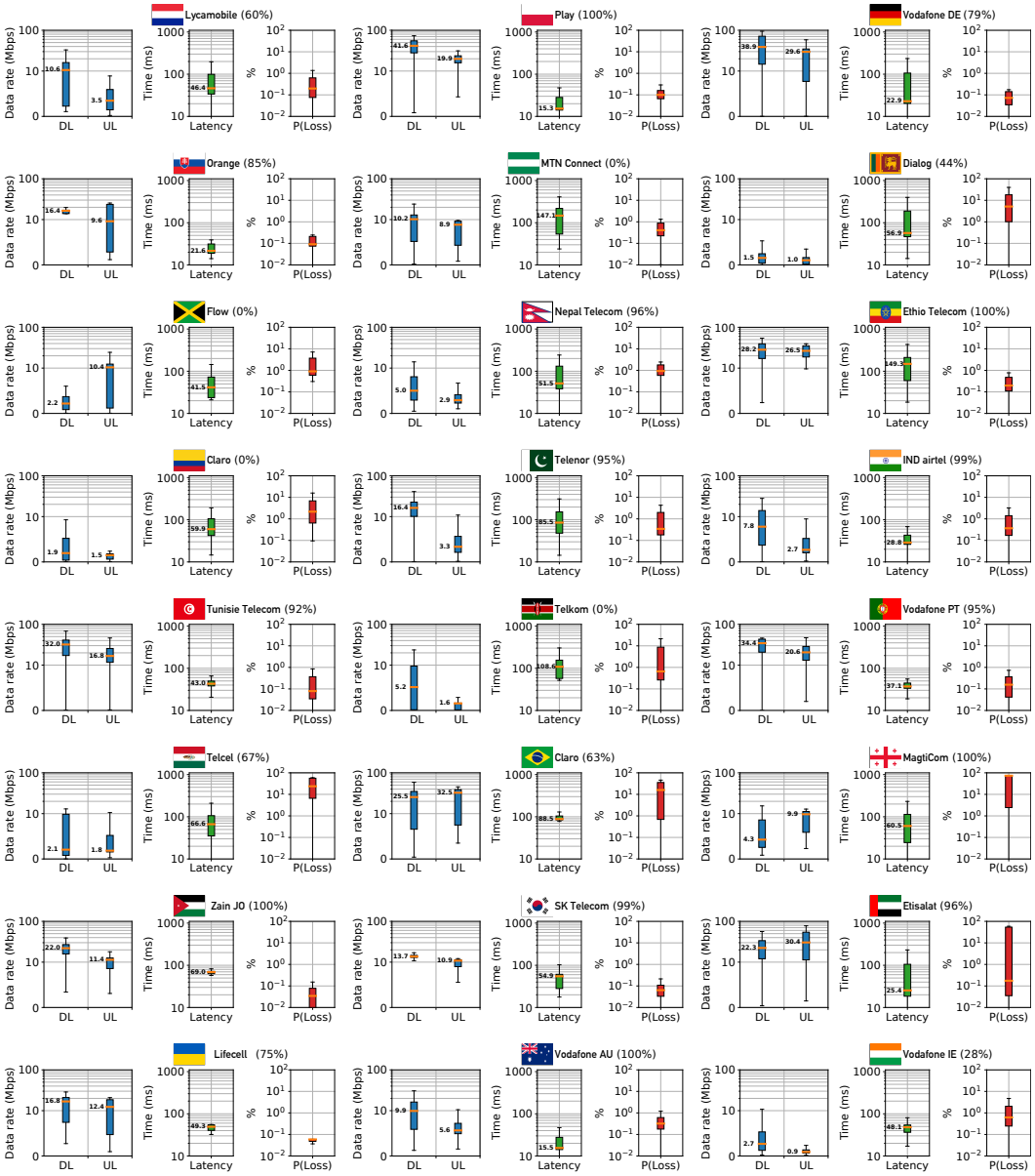
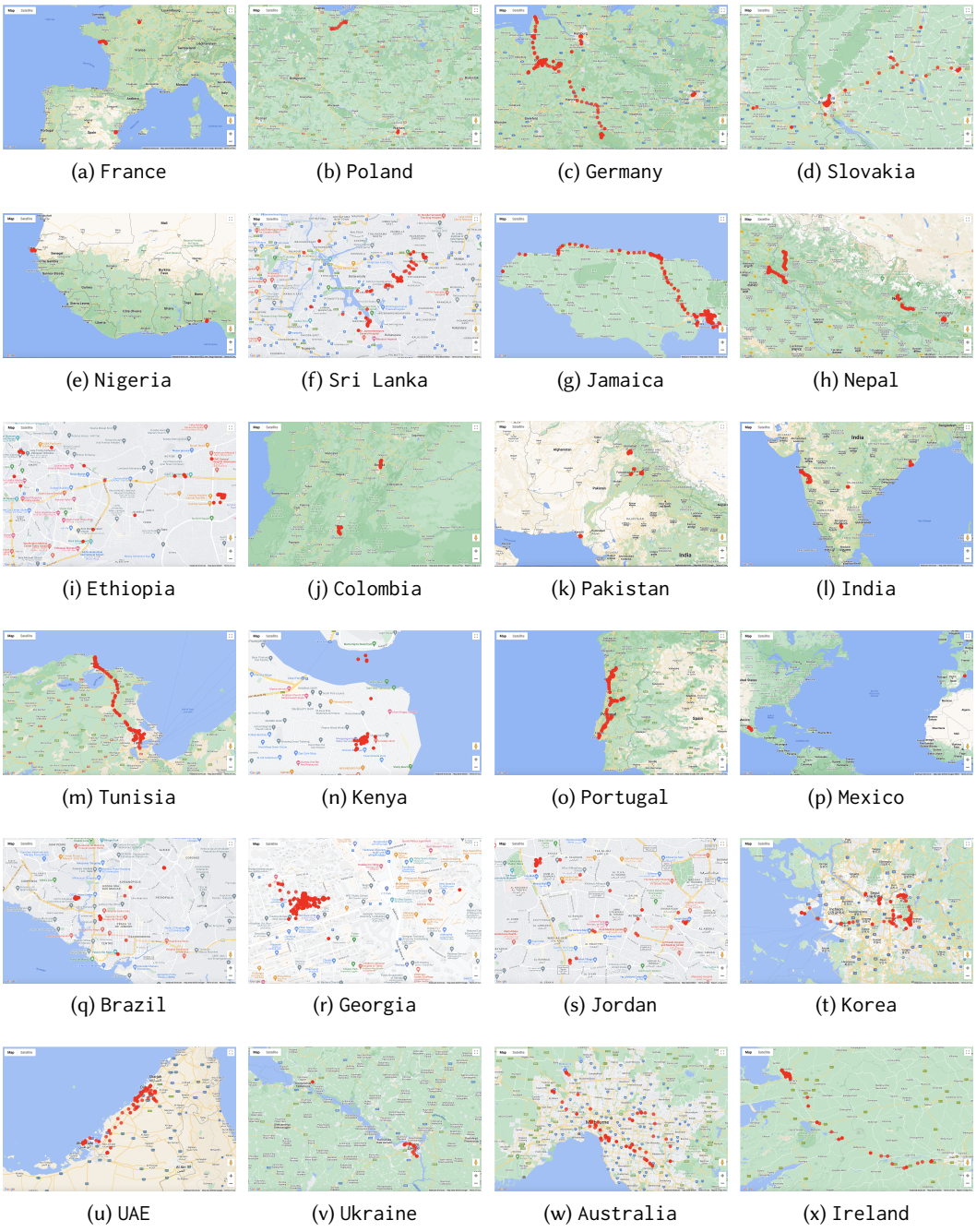


Fig. 15. Overall results for speed test, MTR, and loss probability. The number in parenthesis next to the country flag represents the probability of the phone being on 4G.

Table 2. Measurements metadata

Country (operator)	Speed test	MTR	CDN	YouTube	Web	Distance traveled (km)	Max distance from home (km)	Time at home (days)	Time outside (days)	Data usage
Australia (Vodafone)	242	1614	1486	105	190	225.46	26.01	22.31	0.96	27.3
Brazil (Claro)	35	140	426	33	32	24.55	4.83	20.03	0.32	4.29
Colombia (Claro)	92	608	537	49	56	350.4	294.97	11.47	11.93	5.37
Ethiopia (Ethio Tel)	17	104	117	14	10	81.59	10.7	17.46	1.72	2.94
France (Lycamobile)	25	220	179	20	15	172.42	1058.53	7.06	15.7	2.62
Georgia (MagtiCom)	20	348	345	46	15	33.58	0.79	12.54	0	1.71
Germany (Vodafone)	102	686	602	59	56	1133.27	307.75	20.37	8.37	26.7
India (IND airtel)	325	2033	1946	172	222	1026.58	1365.2	56.01	24.19	28.12
Ireland (Vodafone)	237	1411	1427	187	180	266.64	89.11	18.85	0.87	9.85
Jamaica (Flow)	23	208	123	8	10	858.73	161.48	20.83	6.1	0.52
Jordan (Zain JO)	111	652	666	50	88	57.21	7.06	7.27	0.75	15.75
Kenya (Telkom)	393	2504	2283	215	233	110.56	1.08	34.51	1.19	21.08
Korea (SK Tel)	158	1025	960	121	123	447.98	6567.09	2.21	25.57	16.4
Mexico (Telcel)	9	169	67	6	2	335.27	9509.5	5.8	3.64	0.38
Nepal (Nepal Tel)	45	542	300	39	35	420.58	508.88	43.82	16.52	28
Nigeria (MTN)	108	554	674	82	81	83.42	2444.42	9.76	15.08	9.81
Pakistan (Telenor)	271	1920	1729	184	125	931.04	1121.82	83.51	55.69	34.67
Poland (Play)	271	1943	1632	211	207	41.66	243.31	39.7	4.9	44.86
Portugal (Vodafone)	110	918	669	78	88	953.95	235.93	1.82	19.91	28.16
Slovakia (Orange)	13	154	155	9	13	427.82	75.26	2.76	28.26	1.97
Sri Lanka (Dialog)	105	689	562	59	55	363.71	4.95	1.94	15.43	7.75
Tunisia (Tunisie Tel)	82	519	502	42	56	727.86	66.26	21.63	4.5	16.86
UAE (Etisalat)	73	815	438	55	47	713.85	132.2	21.06	6.42	15.15
Ukraine (LifeCell)	8	57	48	2	4	47.42	121.17	6.56	0.29	0.849

Fig. 16. YouTube quality



(a) France (b) Poland (c) Germany (d) Slovakia

(e) Nigeria (f) Sri Lanka (g) Jamaica (h) Nepal

(i) Ethiopia (j) Colombia (k) Pakistan (l) India

(m) Tunisia (n) Kenya (o) Portugal (p) Mexico

(q) Brazil (r) Georgia (s) Jordan (t) Korea

(u) UAE (v) Ukraine (w) Australia (x) Ireland

Fig. 17. AmiGo users movements' maps