# Mobilyzer: An Open Platform for Principled Mobile Network Measurements

Ashkan Nikravesh[1] Hongyi Yao[1] Shichang Xu[1] David Choffnes[2] Z. Morley Mao[1]
[1]University of Michigan [2]Northeastern University

## Abstract

Mobile Internet availability, performance and reliability have remained stubbornly opaque since the rise of cellular data access. Conducting network measurements can give us insight into user-perceived network conditions, but doing so requires careful consideration of device state and efficient use of scarce resources. Existing approaches address these concerns in ad-hoc ways.

In this work we propose *Mobilyzer*, a platform for conducting mobile network measurement experiments in a principled manner. Our system is designed around three key principles: network measurements from mobile devices require tightly controlled access to the network interface to provide isolation; these measurements can be performed efficiently using a global view of available device resources and experiments; and distributing the platform as a library to existing apps provides the incentives and low barrier to adoption necessary for large-scale deployments. We describe our current design and implementation, and illustrate how it provides measurement isolation for applications, efficiently manages measurement experiments and enables a new class of experiments for the mobile environment.

## 1 Introduction

Given the tremendous growth in cellular data traffic, it is increasingly important to improve the availability, performance and reliability of the mobile Internet. To adequately address these challenges, we ideally would be able to collect network measurement data from any mobile device on any network at any time. With information, users could evaluate the service they are paying for, carriers could study the factors impacting performance and detect problems causing degradation, and application developers could tune and improve their service. Importantly, many of these optimizations can be performed dynamically when data are available in real time.

Despite a need for performance improvement and policy transparency in this space [13, 27], researchers currently still struggle to measure, analyze and optimize mobile networks. Mobile Internet performance characterization is a challenging problem due to a wide variety of factors that interact and jointly affect user experience. For example, the performance that a user receives from an application can depend on the device's available hardware resources, radio's network state, the network access technology used, contention for the wireless medium, distance from nearby cell towers, session initiation overhead, congestion at various gateways and the overhead of executing code at the communicating endpoints. Further, this performance can change over time and as the user moves with the mobile device. Other challenges include resource constraints (data and power) on mobile platforms in addition to direct or indirect interference between measurements.

This problem has not gone unnoticed by researchers, operators and providers. A number of small testbeds and user studies have enabled progress in the face of these challenges [19, 3, 9, 8, 37], but with limited scope, duration, coverage and generality. We argue that previous work suffers from three key limitations that hamper their success. First, these individual solutions do not *scale*: each individual app or measurement platform is inherently limited to the population of participating users running a single piece of software. Second, each solution is *inconsistent* and *inflexible* in the set of network measurements it supports and the contextual information describing the experimental environment, making it difficult to ensure scientific rigor and to merge disparate datasets. Third, these solutions are *uncoordinated* in how they conduct network measurements: multiple apps can wastefully measure the same property independently or, worse, interfere with each other by running measurements at the same time from the same device.

Instead of proliferating apps that conduct independent network measurements in inconsistent ways, we argue that there should be a common measurement service that apps include in their code. Toward this goal, we designed and built *Mobilyzer*, a unified platform for conducting network measurements in the mobile environment. Our system is designed around three key principles: network measurements from mobile devices require tightly controlled access to the network interface to provide isolation, these measurements can be performed efficiently using a global view of available device resources and experiments, and distributing the platform as a library to existing apps provides incentives and low barrier to adoption necessary for large-scale deployments.

*Mobilyzer* provides an API for issuing network measurements using a standard suite of tools, and manages measurement deployment and data collection over a global set of participating devices. Each device runs a measurement scheduler that receives measurement requests and gives experiments explicit control over the context in which a measurement runs. A cloud-based global manager dynamically deploys measurement experiments to devices according to available resources, device properties and prior measurement results.

*Mobilyzer* is currently deployed as a library that can be included in Android apps, and a cloud-based App Engine service for managing measurements across participating devices. We evaluate our deployed system in terms of our design goals. We demonstrate that it effectively provides measurement isolation, and failure to do so can lead to unpredictable and large measurement errors. Further, we show that *Mobilyzer* manages measurement scheduling efficiently, with low delays between requesting a measurement and scheduling it for execution. We also demonstrate how our global manager adapts scheduled measurements based on available device resources and based on information gathered from prior measurements. We provide evidence that *Mobilyzer* is easy to use and significantly reduces development time for apps requiring network measurement, using two *Mobilyzer*-enabled apps as examples.

The rest of the paper is organized as follows. §2 describes the motivation and related work. We describe the *Mobilyzer* design in §3, then highlight key components of our current deployment in §4. We demonstrate in §5 the effectiveness of the library through microbenchmark based evaluation and several usage scenarios enabled through server-based measurement scheduling. We discuss several open issues in §6 and conclude in §7.

## 2 Background and Related Work

Mobile devices are becoming the dominant way for end users to access Internet services; however, mobile network performance is poorly understood by researchers, operators and end users. The research community has limited or no access to network measurements in the mobile environment, hampering dissemination of knowledge and innovation. Operators can gather data directly from their network infrastructure, but often struggle to transform data into actionable knowledge and have limited or no visibility on user-owned mobile devices. Last, users pay a premium for mobile Internet access but lack tools for making informed decisions about carriers based on sound measurements and comparisons.

A variety of existing approaches attempt to shed light on mobile networks. While they improve knowledge in this environment, each prior approach exhibits limitations that we will address with our work.

**Existing research platforms.** Our work shares many of the same goals of successful testbeds such as PlanetLab [29] and RIPE Atlas [32], and our work uses M-Lab [22] servers as targets for many measurement tests. These are general-purpose experimentation platforms that require deployment of infrastructure and do not operate in the mobile environment. These systems are insufficient alone for understanding mobile networks because mobile networks generally use firewall/NATs [41]. Seattle [6] is a general-purpose platform that supports deploying code on Android devices, and it shares many goals with *Mobilyzer*. It does not provide network measurement isolation, but many of its device management and security features can be integrated into *Mobilyzer*.

**View from operators.** Operators and manufacturers such as AT&T and Alcatel-Lucent have deployed in-network monitoring devices that passively capture a detailed view of network flows traversing their mobile infrastructure [11, 4]. Several studies use these passive measurements to understand network traffic generated by subscriber devices, with important applications to traffic engineering, performance characterization and security [14, 31, 43, 4, 40, 21, 33]. However, this approach does not allow the carrier to understand the performance experienced at the mobile device. For example, when the throughput for a network flow suddenly changes, it is difficult to infer from passive measurements alone whether it is due to wireless congestion, signal strength, and/or simply normal application behavior.

Further, the most interesting behavior occurs at or near the edge of mobile networks, making an infrastructure-based deployment costly. For example, Xu et al. point out that monitors located near a cellular provider's core cannot account for detailed user mobility, and adding the infrastructure to support these measurements would require a deployment at least two orders of magnitude larger [42]. Worse, mobile networks are constantly evolving as new 3GPP specs are adopted (7 release versions since 2000), meaning protocols and infrastructure can be displaced within a few years. As a result, the cost of maintaining and upgrading infrastructure-based deployments can be prohibitively high.

**Existing measurement apps.** Motivating the need for a mobile measurement library, a large number of academic, governmental and commercial mobile performance measurement tools have been released in recent years. The FCC released a mobile network measurement app [13] to characterize mobile broadband in the US but the system is closed and data is not publicly available. Several commercial apps measure mobile Internet performance by collecting measurements such as ping latency and throughput [38, 7, 26]. However, because the source code is closed, the methodology is under-specified, and the datasets tightly controlled, it is difficult for the research community and end-users to draw sound conclusions. Further, these tests are generated on-demand by end-users, creating a strong selection bias in the data, as users may be more likely to run tests when they experience problems.

A number of research projects use apps to gather network measurements from hundreds or thousands of mobile devices [36, 16, 30, 20, 35, 34, 23, 24]. It has been shown that such measurements reveal information not visible from passive in-network measurements, such as radio state, signal strength and the interaction between content providers and network infrastructure [44, 42, 18, 36]. Further, these measurements reveal information across carriers, allowing researchers to understand different ISP network policies and performance [41]. Such one-off studies provide a useful snapshot of mobile networks, but they do not support longitudinal studies that inform how mobile network performance, reliability and policy changes over time, nor do they support targeted measurements that are required to understand the underlying reasons for observed network behavior.

Each of the above projects uses a separate codebase, a different set of collected contextual information and different privacy/data sharing policies. As such, experiments are conducted in an ad-hoc manner, with limited or no ability to compare and combine measurements from each limited deployment. As an example pitfall from the current approach, consider the case of interpreting the result of a simple ping measurement. Because mobile device radios enter a low power state during idle periods, a ping measurement that wakes the radio (and experiences additional delay during the wakeup period) can affect a subsequent ping measurement that occurs while the radio is in full power mode. Without proper contextual information, a researcher might falsely attribute network delays to the carrier instead of the device. Worse, if there are two measurement apps running on the same device, one app's measurements can interfere with the other's.

Our work is also motivated by other efforts in building measurement libraries, especially on mobile platforms. For example, Insight [28] offers a platform for instrumenting and measuring application performance, rather than network measurements.

## 3 *Mobilyzer* Design

The goal of *Mobilyzer* is to provide a scalable platform for conducting meaningful network measurements in the mobile environment. It is designed to achieve:

- **Measurement isolation.** The mobile environment poses several unique challenges for controlling how network measurements are isolated from each other. *Mobilyzer* accounts for these properties and gives experimenters explicit control over device state and concurrent network activity during measurement.
- **Global coordination.** Data quota and power are scarce resources for mobile networks, requiring that researchers use more principled approaches than "spray and pray" measurement. With a global view of available network resources and the ability to coordinate measurements from multiple devices, we can coalesce redundant measurements and allow researchers to specify targeted experiments that use device resources efficiently.
- **Standard, easy-to-use measurements.** There is a tension between arbitrary flexibility and standardization in a platform for network measurements. While allowing experimenters to execute arbitrary code within a sandbox provides flexibility, a lack of standards means that these experiments are difficult to incorporate into existing and future datasets. In *Mobilyzer*, we opt for standard measurements, facilitating comparative and longitudinal studies atop our platform.
- **Incentives for adoption.** As a crowdsourcing approach, our platform requires a crowd of mobile users to adopt our platform. We argue that no single app will provide the necessary crowd. Instead, we opt for the "bring your own app" model for deployment, where researchers develop *Mobilyzer*-enabled apps with user incentives in mind. The incentive for researchers to do so is that they can conduct measurements on *any* of the devices in the system (including those not running their app), with measurement quota proportional to the number of devices their app(s) bring to the system. This is analogous to the PlanetLab and RIPE Atlas models, except using a software (instead of hardware) deployment.

The following subsections describe how our *Mobilyzer* design achieves these goals. First we provide an overview of our system, then we describe the key components of our system: the code running inside of each app, the scheduler service running on each device and the cloud-based global manager.

**Nongoals.** Our system has a number of important nongoals. First, we do not provide a "PlanetLab for mobile"; i.e., we do not provide a platform for deploying arbitrary distributed system code on mobile devices. Instead, we focus on the more constrained problem of providing a platform for principled network measurements in lieu of one-off, nonstandard measurement studies. Second, we do not propose any specific incentives for device owners to adopt *Mobilyzer*-enabled apps. Rather, we

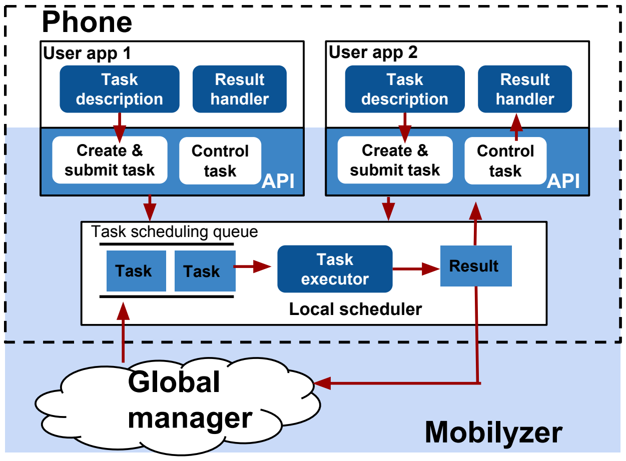


Figure 1: *Mobilyzer* architecture (shaded region). Apps (top) include the *Mobilyzer* library, which provides an API to issue network measurements, and a scheduler service (middle) that provides measurement management and isolation. The scheduler communicates with a global manager (bottom) to fetch measurement requests and report measurement results.

urge experimenters to either develop apps with user incentives in mind, or convince maintainers of existing popular app codebases to include our library. Mobile network diagnosis apps (*e.g.,* speed testing and signal strength maps) are examples of apps that use network measurements and have large user bases. Last, we do not provide a service for arbitrary data-collection from mobile devices; rather, we focus on active and passive network measurements annotated with anonymized contextual information. Collecting arbitrary data from users is a potential privacy risk that we avoid in *Mobilyzer*.

## 3.1 Overview

Figure 1 provides a high-level view of the Mobilyzer platform. It consists of three main components: *Mobilyzer* app library, local measurement scheduler, and cloud-based global manager.

**Measurement Library.** *Mobilyzer* is designed to be easy to use and to integrate into existing apps, while providing incentives for doing so. It is deployed to apps as a library with a well-defined API for issuing/controlling measurements and gathering results locally. This facilitates development of new apps that use network measurements. For example, a new speed test app that includes our library can issue a measurement and retrieve results with under 10 lines of code. Existing apps can incorporate our library to take advantage of our validated measurement tools and server-side infrastructure.

**Local measurement scheduler.** *Mobilyzer* achieves the goal of measurement isolation via the local measurement scheduler. The scheduler listens for measurement requests and ensures that any submitted measurements are conducted in the intended environment (*e.g.,* radio state, whether other apps are using the network and contextual information such as location, signal strength and carrier). Regardless of how many apps on a device use it, *Mobilyzer* ensures that there is always exactly one scheduler running. If apps with multiple versions of scheduler are present, the latest version is guaranteed to be used. The scheduler enforces user-specified limits on resource consumption such as data quota and power.

**Global manager.** The *Mobilyzer* global manager provides coordination for the following services: dynamic measurement deployment, resource cap enforcement, and data collection. Experimenters use the global manager to deploy measurement experiments to devices. For example, an experiment may contain a number of network measurements that should be conditionally deployed and executed according to device properties. The global manager deploys these measurements to devices based on device information reported periodically, and ensures that measurements are not deployed to devices that have exceeded user-specified resource caps. The manager maintains a datastore of anonymized data collected from all devices and makes this available for interactive measurement experiments and for offline analysis. The data collection and scheduling support at the global manager enable dynamically triggered measurements based on observed network behavior, achieving a feedback loop of measurement, analysis, and updated measurements.

In addition to reducing development time for apps requiring network measurements, *Mobilyzer* provides incentives in terms of access to run measurements on any *Mobilyzer*-enabled device. This access is subject to available resources on those devices, and in cases of oversubscription, resources are allocated to experimenters in proportion to the number of devices they brought into the system.

## 3.2 Measurement Library and API

A key design principle for *Mobilyzer* is that the platform should remove barriers to widespread adoption so that it facilitates a large-scale deployment. As such, we chose to implement *Mobilyzer* as a network measurement library that app developers include in their code. This code provides an API for issuing measurements using a standard suite of tools (Table 1), and a device-wide measurement scheduler that the API communicates with.

The measurement model supported by *Mobilyzer* represents a trade-off between complete flexibility to run arbitrary code and complete safety in that all resources consumed by a measurement can be predicted in advance. Specifically, *Mobilyzer* supports a small set of commonly used *measurement primitives*, e.g., ping,

| API Function | Description |
|---|---|
| `MeasurementTask` **`createTask`**`(TaskType type, Date startTime, Date endTime, double intervalSec, long count, long priority, Map<String,String> params)` | Create a task with the specified input parameters including task execution frequency task priority, etc. |
| `void` **`submitTask`**`( MeasurementTask task )` | Submit the task to the scheduler. |
| `void` **`cancelTask`**`(String taskId)` | Cancel the submitted task, only allowed by the application that created this task. |
| `void` **`setBatteryThreshold`**`(int threshold)` | Set the battery threshold for check-in and running the global manager scheduled tasks. |
| `void` **`setCheckinInterval`**`(long interval)` | Set how frequently the scheduler checks in. |
| `void` **`setDataUsage`**`(DataUsageProfile profile)` | Set a limit for *Mobilyzer*'s cellular data usage. |

Table 1: *Mobilyzer* key API functions to support issuing and controlling measurement tasks.

traceroute, DNS lookup, HTTP get and throughput tests. These measurements can be performed independently in isolation, or they can be chained (do a DNS lookup for google.com, then ping the IP address) or executed in parallel (e.g., ping google.com during a TCP throughput test) in arbitrary ways.

Using this model, the amount of data and power consumed by each task (simple or complex) is predictable with few exceptions (e.g., downloading an arbitrary Web page) that can be mitigated via scheduler-enforced constraints on the data and power consumption of a task. In particular, each measurement is assigned a maximum amount of resources it can consume and the scheduler (below) enforces these limits.

Note that *Mobilyzer* does not support running arbitrary code on mobile devices as done by platforms such as PlanetLab and Seattle. A key challenge for *Mobilyzer* in this environment is how to provide measurement isolation and how to predict resource consumption before deploying experiments. Exploring how to support this feature in *Mobilyzer*, potentially using existing platforms such as Seattle, is a topic of future work.

## 3.3 Local Measurement Scheduler

The local measurement scheduler is a service running on a device that manages the execution of measurement tasks. It can be implemented in the operating system or run as a user-level background service. The scheduler provides a measurement execution environment that can enforce network isolation from other measurements and apps running on the device. It also enforces resource-usage, priority, and fairness constraints.

**Task priorities.** The scheduler supports measurement task priorities with pre-emption. In short, it executes tasks with highest priority first and will pre-empt ongoing measurements if they are of lower priority. Certain measurement tasks cannot produce correct results if pre-empted (*e.g.,* a ping task pre-empted between sending an echo request and receiving a response), so pre-emption will lead to wasted measurement. To balance the trade-off between this waste and scheduler responsiveness to high-priority tasks, the scheduler waits for a short time (*e.g.,* one second) before pre-empting a measurement.

This is sufficient time for a ping measurement to complete, but not necessarily for other tasks such as a traceroute. To minimize wasted measurement resources for pre-empted tasks, we define a `pause()` method that signals to measurement tasks that they should save their state due to an imminent pre-emption. In the case of traceroute, this means that the traceroute measurement can save its current progress and continue to measure a path once it regains access to the network interface.

Not all measurements can (or should) be pre-emptible. In addition to the traceroute task mentioned above, we have also implemented a pre-emptible long-running task for inferring RRC state timers on mobile devices [12, 17]. Measurements like DNS lookup and ping are not pre-emptible (i.e., they are killed at pre-emption time) because they are short-lived tasks with minimal (or no) state to cache. Even for traceroute, paths may have changed after pre-emption. In general, network measurement tasks (e.g., throughput measurement) to characterize time-varying properties of the network cannot benefit from saved state when pre-empted; this is in contrast to measurements of more stable properties (e.g., RRC state machine and NAT policies).

**Resource constraints and fairness.** The measurement scheduler accounts for how much data and power is consumed by network measurements and ensures that they stay within user-specified limits. If there is insufficient quota, measurement requests are cancelled until the constrained resource is replenished. For energy, this happens when a device is recharged; for data plans, this is typically done according to the billing cycle.

All measurement tasks generated by apps running on a device are assigned the same high priority value. To ensure fairness among multiple apps on the same device competing for resources, we ensure that each app receives an equal share of the constrained resource.

**Fault tolerance.** In the mobile environment, processes are subject to being killed due to device restarts, as well as battery and memory exhaustion [2]. Thus, the scheduler must be resilient to unexpected termination.

There are two key components to ensuring that the measurement application is fault-tolerant. The first is robust scheduling. Tasks that are low-priority, long-running and periodic can suffer from indefinite postponement when scheduler state is lost upon restart. *Mobilyzer* supports saving key scheduler state to persistent storage in order to ensure that tasks will run as often as intended even in cases where the application does not run continuously for days.

The second component is task persistence. The scheduler reports measurement results to the global manager (below) periodically, so it is possible that the scheduler is killed after measurement completion but before the results are uploaded, leading wasted measurements. To address this problem, the scheduler saves measurement results to persistent storage after completion. Similarly, our platform can support task persistence by given them access to a small, fixed amount of persistent storage for saving intermediate results.

### 3.4 Global Manager

The *Mobilyzer* manager maintains a global view of measurement resources, efficiently dispatches experiment tasks to available devices and coordinates measurements from multiple devices to meet various experiment dependencies. Atop the manager is a Web interface that allows experimenters to specify measurement experiments that are deployed to devices.

Specifically, researchers can submit a measurement schedule, along with information about the properties of devices that should participate in the experiment (e.g., location, device type, mobile provider). Devices periodically check in with the manager and receive a list of experiments to run. When complete, devices report the results of the measurements to the manager.

The manager ensures: (i) no experiment uses more than its fair share of available measurement capacity across *Mobilyzer* devices; (ii) experiments are scheduled in a way that maximizes the likelihood that the targeted devices will complete their measurements during the period of interest; and (iii) it enforces limits that prevent harm to the network or hosts (*e.g.,* via a DDoS).

**Fairness under contention.** In a successful *Mobilyzer* deployment, there are likely to be cases where measurement requests exceed available measurement resources. In periods of contention, we must dispatch measurement tasks according to some fairness metric.

We assign measurement tasks to devices such that the fraction of assigned tasks for an experimenter is proportional to the number of measurement-enabled devices that the experimenter has contributed to *Mobilyzer*. When measurement demand from an experimenter is lower than their fair share, we backfill available resources from remaining pending tasks to ensure that our system is work conserving. Similar to previous work [5, 25], we use an auction-based approach as a building block to provide such fairness.

**Availability prediction.** Available devices in *Mobilyzer* are subject to high churn and mobility, meaning that simply deploying an experiment to some random fraction of devices may lead to a low rate of successful measurements. To improve this rate, the global manager incorporates deeper knowledge of experiment dependencies, prediction for resource availability and accounting for failures due to issues such as disconnections and loss of power. For example, we use prediction to prevent wasted measurement resources by preventing experiments from being dispatched if they are likely to fail (*e.g.,* due to unavailable measurement quota or due to measurement preconditions not being met on the device).

**Live measurement experiments.** With a global view of measurement results gathered from the system, *Mobilyzer* allows researchers to specify measurement experiments that are driven by results from ongoing measurements; i.e., experiments that cannot be specified a priori. For example, in §5.4.1, we demonstrate how this feature enables us to understand the quality of CDN redirections in the mobile environment.

**Preventing harm.** The global manager can account for all measurement activity in the system and ensure that no host or network is overloaded by measurement traffic. In addition, we support blacklists for measurement targets, much like other measurement systems such as iPlane, but with more awareness for resources on mobile platforms.

## 4 Current Deployment

*Mobilyzer* is currently implemented and deployed on Android devices communicating with a Google App Engine global manager. The *Mobilyzer* system as described in this paper has been deployed for one month, though an early prototype has been collecting measurement data since June, 2012. The source code for the system is public and released under an open-source license.[1] The following sections describe several implementation details and challenges specific to our current deployment.

### 4.1 Measurement Library and API

The *Mobilyzer* measurement library is implemented as Android code that is added by reference to an existing app's source code. It requires only three simple steps to create a *Mobilyzer*-enabled app using the Eclipse development environment.

**Basic Measurements Supported.** *Mobilyzer* supports both passive, contextual measurements and active net-

[1] We have omitted links for double-blind review.

work measurements. Currently, the active measurements supported are DNS lookups (with arbitrary targets), ping, traceroute, and a TCP throughput and UDP burst test.

*Mobilyzer* also collects passive measurements on both network performance and device state, and associates a set of passive measurements with every active test run. It collects the signal strength (RSSI values) and the device's battery level, battery charging state, coarse-grained location, the network technology, the total number of bytes and packets sent and received by the device, as well as static context information such as the carrier, OS, and support for IPv6.

Our set of measurement primitives and monitored device properties do not require special privileges; however, we will support measurements that require rooted phones (if available).

**Supporting complex measurements.** Simple tests such as pings and throughput measurements have limited duration and thus are easy for the local scheduler to manage. Long-running tasks, such as the one described below, motivate the need for additional scheduler features to ensure measurements complete successfully.

Mobile devices change between different power states, called RRC states, in response to network traffic [12, 17]. Identifying how long devices stay in each power state entails sending a large number of individual packets with very long gaps between them and inferring power states based on observed packet delivery delays.

This type of measurement poses two key challenges. First, it is a long-running, low-rate measurement that should run only when connected to a cellular network. With no other tasks running on the device, the test can easily take half an hour. Given our pre-emptive priority scheduler, the RRC inference task will either block other tasks for long durations or will be constantly interrupted by higher priority tasks. The latter situation can lead to a large volume of wasted measurement bandwidth (as interrupted measurements must be discarded and restarted), and with sufficient interruptions the RRC task may not have a chance to complete.

Second, this task requires complete isolation from other network activity. Any concurrent traffic will alter the results of this test by changing the radio power state. To detect whether the network isolation property holds, *Mobilyzer* collects information about device-wide background traffic through the *proc* file system and discards results taken when there is background traffic.

These challenges motivate the following approach for the RRC state task. First, it runs with low priority so it does not block other measurements for unreasonable periods. Second, because it may be frequently pre-empted or unable to run due to the device using WiFi, we support suspending and resuming this task. Third, we upload partial measurement results when the task fails to complete, so the data collected is not wasted.

## 4.2 Local Measurement Scheduler

The scheduler is implemented using an Android service, which guarantees that at most one scheduler is present regardless of the number of *Mobilyzer*-enabled apps running on a device. The library communicates with the scheduler service via IPC calls.

**Task management.** The scheduler maintains a priority queue of pending tasks and ensures that exactly one task (simple or compound) executes at a time. It also collects device contextual information periodically during measurement and makes this information available in the measurement result.

When launching a task, the scheduler creates a new thread for the task, then monitors its progress. If a new task with higher priority enters the queue, the scheduler attempts to pause the task. If the measurement supports pausing, it is placed in the queue of waiting measurements; otherwise it is forcibly killed and the higher priority task is dispatched.

For measurements that come from the global manager, *Mobilyzer* enforces resource constraints specified by the user to ensure that measurements do not consume too much data or battery. The scheduler tracks data and power usage for the measurements it conducts and discontinues measurements during periods when there is insufficient quota. These limits are not enforced for measurements generated by *Mobilyzer*-enabled apps, because we do not want to interfere with app functionality. For example, when a user triggers a measurement from an app that provides information about TCP throughput speeds, we allow the test to proceed regardless of limits.

**Forward compatibility.** Our library-based deployment model means that different *Mobilyzer*-enabled apps running on the same device may use different versions of the measurement library and scheduler. We want to ensure that all apps on the device bind to the *newest* version of the scheduler service as soon as it is installed by an app. A key challenge for providing this functionality is that by default on Android, apps will bind only to the scheduler of the app that is first installed, which is unlikely to be the latest version.

We address this as follows. First, we exploit the Android priority tag in the bind Intent when declaring the scheduler in the manifest file. The scheduler with the highest priority is bound if there is no other bound scheduler, so we increment the priority value with each new *Mobilyzer* version. However, if an older scheduler has already been bound, apps already bound to an older version do not switch until the device is rebooted. To address this issue, we deliver the scheduler version information via the start Intent. When the scheduler receives a start Intent, it compares its version with the one in the intent and terminates if it see a newer scheduler version.

### 4.3 Global Manager

The *Mobilyzer* manager is currently implemented using a Google App Engine (GAE) instance, providing a highly scalable platform for managing thousands or millions of devices. The manager currently uses a pull model for experiment deployment, where devices check in with the manager periodically to retrieve updated experiments and to update their contextual information (coarse location, remaining battery, access technology). We are investigating how to incorporate Google Cloud Management services to enable a push-based model for experiments with tighter timing constraints.

Experimenters currently can specify simple measurement schedules using an interface that permits control over measurement parameters, experiment durations and periodicity. We have also developed several dynamic measurement experiments as described in Section 5.4, where the measurements issued to a device depend on the results of prior measurements (from potentially many other devices).

The manager deploys measurements based on contextual information gathered from devices. For example, the manager will reduce the measurement rate for periodic tasks that would otherwise exceed device quota. Similarly, measurement tasks for devices in a specific geographic region are not deployed to devices outside that region.

## 5 Evaluation

We now evaluate *Mobilyzer* in terms of deployment experience, performance, and applications.

### 5.1 Deployment Experience

One key advantage of *Mobilyzer* is that apps requiring network measurements are easier to write and maintain. Our platform provides validated measurement code, prevents interference from other *Mobilyzer*-enabled apps, collects and stores measurement data and utilizes existing infrastructure (via M-Lab) for bandwidth testing. We believe that by separating measurement management and data collection code from app development, *Mobilyzer* helps researchers focus on interesting measurement experiments and compelling incentives for user adoption.

Table 2 lists the lines of code (LoC) used for network measurements in three popular measurement apps. In our reference app (anonymized for double-blind review), 80% of the code was for measurement. Using a library-based model, we simplify the app codebase, making it easier to focus on UI and other elements to encourage user adoption. With *Mobilyzer* one can issue a measurement task and retrieve the results with fewer than 10 LoC.

In addition to porting the reference app to use *Mobilyzer*, we made our library available to the developers of MySpeedTest, a throughput-testing app. The experience from the main developer was overall quite positive, and identified cases where instructions for use were unclear. While it took several e-mail exchanges to clarify these issues, the developer reported that writing the code to port MySpeedTest to *Mobilyzer* took “about an hour or less”. We are currently in discussions with other researchers about integrating *Mobilyzer* into their apps.

| Measurement App | Measurement LoC |
|---|---|
| FCC SpeedTest [13] | 12550 |
| MySpeedTest [23] | 9545 |
| Reference App | 8976 |

Table 2: Lines of code (LoC) for open-source measurement apps. By integrating *Mobilyzer* into existing apps, developers can save thousands of lines of code.

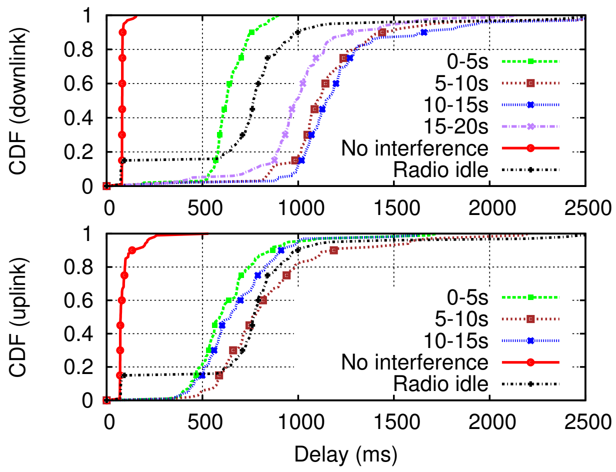


Figure 2: Impact of throughput measurements (broken down into 3–4 time periods) and radio state on measured RTT. Results vary with radio state, and upstream or downstream cross-traffic with varying throughput. *Mobilyzer* provides strict control over these kinds of dependencies.

### 5.2 Measurement Isolation

An important feature of *Mobilyzer* is that it schedules measurements to provide applications with control over if and when measurements are run in isolation. We now use this feature to demonstrate the value that isolation brings. In particular, we use a compound measurement task that consists of a TCP throughput test run in parallel with a ping measurement. We vary the start time for the ping measurement such that it occurs before, during, and after the throughput test and plot the results. We also control whether the cellular radio is in a low-power state before measurement. Each experiment is repeated 40 times; we plot a CDF of ping latencies for each configuration. We omit throughput results because they are unaffected by ping measurement cross traffic.

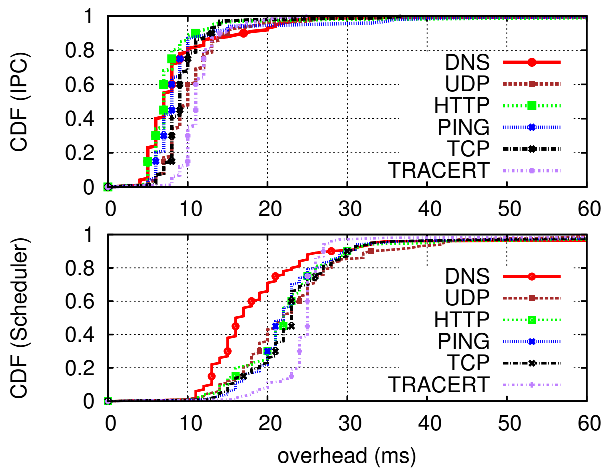


Figure 4: IPC and schedule overhead, 1 app.

Figure 2 shows that ping measurements are significantly affected by cross traffic, and the difference in measured latency compared to the case with no interference can be large as hundreds of milliseconds or seconds. The additional delay, presumably from queuing behind the TCP flow from the throughput test, also varies depending on how much cross traffic occurs. As a result, interference from cross traffic renders the latency measurement meaningless. Note that the downlink inference is more severe likely due to higher throughput (2.5 Mbps down vs 0.35 Mbps uplink), which results in more queuing both inside the network and on the device.

The figures also show differences in measured latencies when the radio is active compared to when it is idle before measurement. Similar to the above scenario, not accounting for this effect can cause misleading or incorrect conclusions. With *Mobilyzer*, experimenters can tightly control the impact of these sources of noise in measurement experiments.

## 5.3 Microbenchmarks

This section presents results from controlled experiments evaluating *Mobilyzer* overhead in terms of measurement-scheduling delay, power usage, and data consumption.

### 5.3.1 Scheduling Delays

The scheduling delay introduced by *Mobilyzer* consists of the delay from using IPC (interprocess communication) between an app and the scheduler, and the delay introduced by the scheduler. As we demonstrate, these delays are reasonably low for all measurements and under significant load.

The IPC delay (Fig. 3) is the sum of (i) the delay between when the client submits a task and when the server receives it (IPC part I in the figure), and (ii) the delay between when the scheduler sends the result to the client, and the client receives that result (IPC part II).

We use an HTC One (Android 4.1.1, using LTE) as our test device to run several measurement tasks for characterizing the IPC overhead. Each task is run 100 times. Fig. 4 presents a CDF of the delay; for all tasks, the maximum delay is below 100ms, while most delays are within 20ms. These values match the performance analysis for IPC latency using Intents [15]. Note that this delay affects the time until a measurement runs but does not interfere with measurement execution.

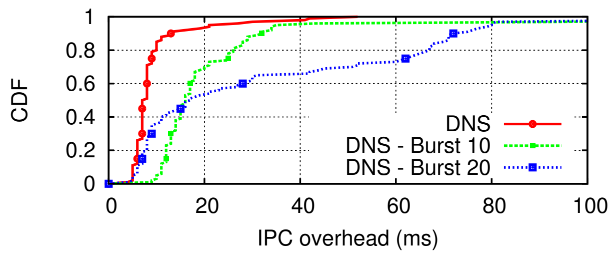


Figure 5: IPC overhead for different burst size for DNS.

Note that the IPC overhead increases when many tasks are submitted back to back, due to the way Android implements Intent-based IPC. Specifically, there are two IPCs for each Intent-based IPC – one from sender to the Intent manager, and then one from the manager to the receiver [15].

To test the impact of this, we submit bursts of 1, 10 and 20 tasks with 15 ms between each burst. If several IPCs are sent to the Intent manager at once, there may be a delay in redirecting them to the receiving app. Fig. 5 shows the IPC overhead under high load for the DNS task (other tasks exhibit a similar pattern). Although larger burst sizes lead to longer delays, with a burst of 10 the delay is mostly within 50 ms. For a burst of 20, over 90% are below 100 ms, which we believe is acceptable for scheduling measurement tasks. It is unlikely that user-generated activity will create such a large burst of measurements, so we do not expect these delays to affect user-perceived responsiveness.

We similarly measure the scheduling delay introduced by *Mobilyzer*, *i.e.,* the delays in starting the measurement and sending the result (schedule overhead part I and II in Fig. 3). The scheduling delay distribution for a single client is shown in Fig. 4. It is slightly higher than the IPC overhead, but still on the order of tens of milliseconds. Again, we believe this is acceptable for scheduling and reporting results from measurement tasks.

### 5.3.2 Power Usage

We now estimate the power consumed by *Mobilyzer*. In general, it is difficult to distinguish power consumption of our library from the application that runs it. To address this, we measure the power consumption of an app using the *Mobilyzer* service as it runs in the background executing server scheduled measurement tasks and compare it to power consumed by the same app before integrating

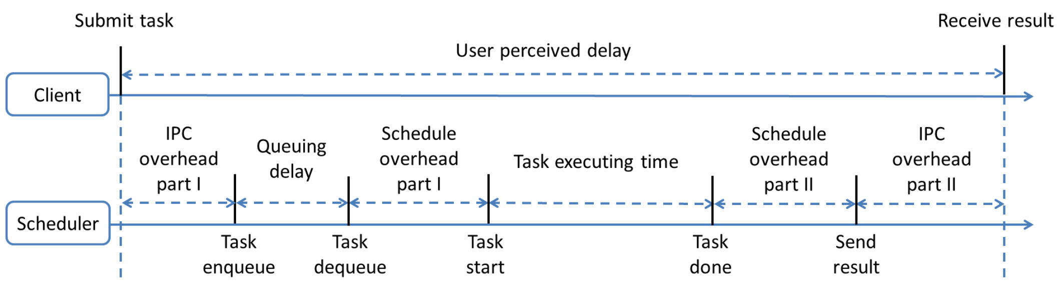


Figure 3: Overhead imposed by *Mobilyzer* in scheduling a measurement task.

| Task | Power consumption under WiFi (mAh) | Power consumption under LTE (mAh) |
|---|---|---|
| DNS | 0.03 (0.01) | 0.09 (0.01) |
| HTTP | 0.05 (0.01) | 0.12 (0.01) |
| UDP (Down) | 0.06 (0.02) | 0.17 (0.02) |
| UDP (Up) | 0.08 (0.03) | 0.17 (0.04) |
| PING | 0.21 (0.01) | 0.52 (0.02) |
| TCP (Down) | 0.11 (0.28) | 2.88 (0.24) |
| TCP (Up) | 1.33 (0.04) | 2.36 (0.20) |
| Traceroute | 0.75 (0.25) | 2.34 (0.02) |
| Sum(unbatched) | 3.51 (0.38) | 8.66 (0.32) |
| Sum(batched) | 3.75 (0.61) | 5.06 (0.25) |

Table 3: Power consumption of *Mobilyzer*: each cell contains the average, then the standard deviation in parentheses.

*Mobilyzer*, with the overall functionality kept the same.

We measure the power consumption of several measurement tasks using a Samsung Galaxy Note 3 as the test device, and a Monsoon power monitor [1] to measure the device power usage.We run each task 5 times with a 15-second delay between tasks to determine the power drain in isolation (unbatched). Additionally, we run a batched task consisting of all 8 tasks 5 times, and measure the power consumption during the entire measurement period to better reflect that most tasks in *Mobilyzer* run in batches. For comparison, we also sum up the average power consumption for all 8 tasks, denoted as Sum(unbatched) based on individual experiments. All experiments are repeated 3 times to identify possible outliers, and we record the average power consumption. Table 3 presents the results when measuring over WiFi and LTE.

Of the unbatched experiments, we can see that the TCP throughput test has the highest power consumption. This is consistent with prior results showing that power drain is roughly linear with the throughput and duration that the network interface is in a high power state [17]. The power drain from remaining tasks is proportional to the measurement duration because their throughput is low. Note that traceroute has higher power consumption under LTE than under WiFi, since it sends a large number of ICMP packets with an interval of roughly 0.5s. While on LTE, the device will stay in the high power state between these packets.

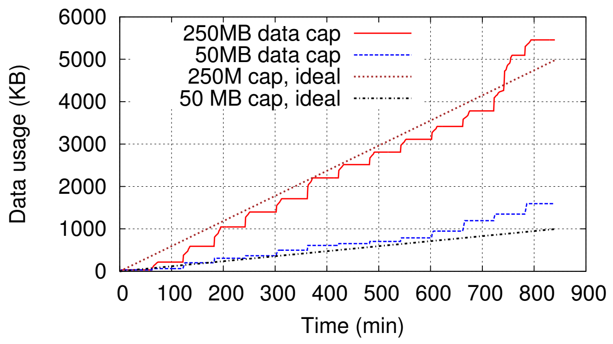


Figure 6: Data usage under different data caps.

For batched tasks, we found that on WiFi the power consumption for batched and unbatched tasks are similar. However, on LTE the total power consumption of the batched measurements is much smaller than that of the sum of the individual measurements, by 41.5%. This is because of the tail energy effect on LTE, where the device remains in a high power radio state for several seconds after a network transmission. In each individual task, there is a contribution from the tail time of an average of 48.2% to the total energy consumed [17]. When the tasks are batched, there is only one tail time, demonstrating the effect of batching in saving energy in cellular networks [17].

While these micro benchmarks help us understand the power of individual measurements, it does not indicate the relative cost compared to other services running on devices. We evaluate this using the built-in Android battery consumption interface on a Samsung Galaxy S4 device actively running a *Mobilyzer*-enabled app with a 250 MB monthly measurement quota. We find that the power consumed by *Mobilyzer* is nearly identical to the Android OS itself, with each comprising about 5% of the total energy drain. We believe this power consumption to be reasonably low for most users.

### 5.3.3 Data Usage

This section evaluates the effectiveness of adaptive measurement scheduling in response to per-device caps of *Mobilyzer* data consumption. Our system consumes data quota from (i) fetching measurement requests from the global manager, (ii) uploading measurement results, and (iii) running the measurement tasks. The last category constitutes the vast majority of data consumption, so we set the frequency of periodic measurements according to each device's data cap. We now demonstrate how well this works in practice.

For this experiment, we monitor the data consumption on a HTC One device by reading the proc files under `/proc/uid_stat/<app uid>` once per minute to monitor the app-specific data usage for *Mobilyzer*. Figure 6 shows *Mobilyzer*'s data usage on a cellular network during 14 hours for a 250 MB and 50 MB data cap, along with the corresponding ideal data consumption. The server adjusts the base measurement frequency of each task based on estimates of the data consumed by each task and the given data cap. Data is consumed at a slightly higher rate than expected, as the server estimates are not precise. For more precise data consumption control, a client-side data monitor measures all data consumed by *Mobilyzer* and stops running server scheduled tasks as soon as the limit is reached. Client-side data monitoring has been developed but is not deployed in *Mobilyzer* at the time of submission.

### 5.4 Server Scheduling

The *Mobilyzer* global scheduler enables dynamic, interactive measurement experiments where tasks assigned to devices vary in response to results reported from prior measurements. We present two applications of this feature: measuring the *CDN redirection effectiveness* of major CDNs, and *network diagnosis with adaptive scheduling*, which dynamically schedules diagnosis measurements in response to observed performance issues.

#### 5.4.1 CDN Redirection Effectiveness

In previous work [39], Su et al. used PlanetLab-based experiments to show that CDNs typically send clients to replica servers along low latency paths, instead of optimizing for other factors such as load balancing. This requires measuring paths not only to replica servers returned by a DNS redirection, but also testing the latency to other replica servers that could have been used instead. Using a dynamic measurement experiment, we now repeat this study for the mobile environment to test the extent to which this holds true.

Specifically, we select DNS names served by five large CDNs (Akamai, LimeLight, Google, Amazon CloudFront, and EdgeCast) and measure the latency to the servers that mobile clients are directed toward via DNS lookups. In addition, we measure the latency to servers (9 servers selected randomly, one per /24 prefix) that *other* devices are directed toward.

For each round of measurement, we find the delay difference between the lowest-latency server and the one returned by the DNS lookup to determine how optimal the mapping is. In addition, we sort the latencies to determine the *rank* of the server. For example, if the server returned by the CDN is the second-fastest of ten, its rank is 2.

We summarize our rank results in Figure 7 for 30 devices which used either WiFi or cellular. Each data point represents the ranking result for one round of measurement. Note that due to our small sample size, we cannot draw strong conclusions about global results. Rather, these measurements demonstrate CDN performance for a small set of *Mobilyzer* users.

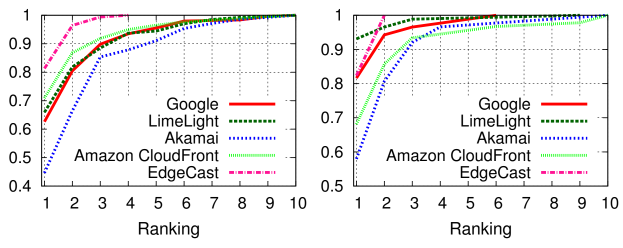
The figure shows that most CDNs (*e.g.,* Akamai) pick a relatively good server (low rank value) most of the time for both WiFi and cellular users. Interestingly, the rank is slightly better for the cellular users in our dataset. We suspect this is due to the limited number of cellular peering points with the public Internet, making it easier for a CDN to identify a nearby replica.

Figure 7 shows that the optimal server is not selected all the time, but does not indicate whether that has a significant impact on client-perceived performance. To determine this, we plot the latency difference between the lowest-latency CDN replica and the CDN-selected replica for the same devices in each round of ping measurements. In more than 40% of cases, the latency to the CDN-selected server is optimal. Figure 8 plots the latency difference for cases where the CDN-selected replica is not optimal. We find that the vast majority of these cases lead to a latency difference of 10s of ms, which is relatively small for Web workloads (and consistent with prior work).

In the worst 10% of cases, however, the latency difference can be as high as 100 ms and 50 ms for WiFi and cellular users, respectively. Interestingly, the worst-case performance difference over WiFi is worse than in cellular. indicating that the diversity of path performance to various CDN replicas is greater in the WiFi cases than the cellular network we measured.

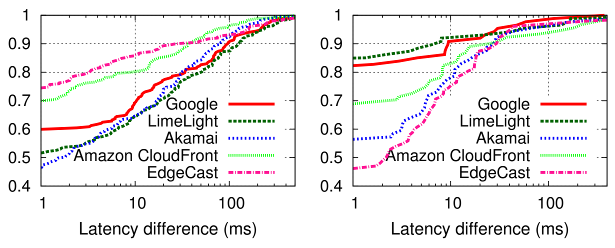
#### 5.4.2 Network Diagnosis with Adaptive Scheduling

Dynamic scheduling not only enables new measurements in the mobile environment, it also provides an opportunity to improve measurement efficiency. For example, consider a measurement experiment that measures performance periodically and issues diagnosis measurements in response to anomalous conditions, *e.g.,* to isolate whether there is a problem with a specific server, endpoint or network. Without a priori knowledge about when and where network problems will manifest, the only way to ensure diagnosis information is always available is to issue diagnosis measurements even when there is no problem detected.



(a) WiFi (b) Cellular

Figure 7: CDF of Rank of the ping latency for the CDN-selected server compared to other random CDN IPs for (a) 25 devices with WiFi access and (b) 7 devices with AT&T cellular access.



(a) WiFi (b) Cellular

Figure 8: CDF of Ping latency difference between the CDN-selected server and the random CDN IP with lowest latency for (a) 25 devices with WiFi access and (b) 7 devices with AT&T cellular access.

We now use a simple example to show how *Mobilyzer* allows us to issue diagnosis measurements on demand in response to observed problems and estimate the bandwidth savings from doing so. In particular, each device performs local detection of anomalous latency to CDN servers using a moving average (as done in NEWS [10]), then conduct a traceroute to locate the region of the network affected by the problem. Using dynamic scheduling, *Mobilyzer* then dispatches diagnosis measurements to other devices on the same network, and to devices not on the same network to determine the scope the problem.

We run this experiment with 70 users over 12 days to see how frequently and under what conditions *Mobilyzer* schedules new measurement tasks, then compare this with the case where every device conducts a diagnosis task in addition to a measurement task. Our results show that diagnosis tasks are scheduled for only 3% of cases.

Even with a task as low-cost as a traceroute measurement (1.5KB), this can lead to substantial savings in system-wide power and data quota resources. For example, even with only 1,000 users, this amounts to over 1GB of bandwidth and up to 500mAh of power.

## 6 Discussion

This paper makes the case for network measurement as a service in the mobile environment, and demonstrates how we addressed several challenges toward making this service practical and efficient. We now discuss important challenges that are outside the focus of this paper.

**User incentives.** Our solution relies on users to install software that is instrumented with network measurements, raising the question of whether there are proper incentives. One key advantage of our design is that *Mobilyzer* does not rely on a single app; rather, it provides a measurement library that can be included in any number of apps. By shifting the focus from a single popular app to multiple apps with varying degrees of popularity in different regions, we can leverage a variety of incentives via apps providing free, desirable services. For example, we can provide information regarding whether users are getting the service they are paying for, details about how providers are shaping or interfering with their traffic and how other apps are affecting battery life, data-quota consumption and overall performance. We note that several apps in this space [38, 28] already boast user populations in the hundreds of thousands or millions; simply integrating our library into an existing popular app could instantly bring critical mass to *Mobilyzer*.

**Privacy.** Any system that collects data from users must carefully protect privacy. This study is IRB-approved and participating users are consented before participating via an in-app dialog. Users may report data via a Google account, which allows them to view and delete their data at any time, or they may report data anonymously. We strip all user information, including UUID, from the data we gather; further, we coarsen the location granularity to one square kilometer.

**Security.** Like Dasu and Seattle, *Mobilyzer* network activity can be controlled by a centralized service. We currently use existing PKI techniques such as signed app jar files and HTTPS communication with the global manager to secure communication and authenticate entities. Our local scheduler also enforces hard limits on the device to prevent resource exhaustion (*e.g.,* battery drain). We are exploring opportunities to incorporate other security features from Dasu and Seattle.

## 7 Conclusion

In this paper, we make the case for a unified platform for conducting network measurements from mobile devices. Our system, *Mobilyzer*, provides (i) network isolation to ensure valid measurement results, (ii) contextual information to ensure proper scheduling of measurements and interpretation of results, (iii) a global view of available resources to facilitate dynamic, distributed experiments and (iv) a deployment model with incentives for experimenters and app developers. We showed that our system is efficient, easy to use and provides an environment to support new measurement experiments in the mobile environment. As part of our future work, we are designing an interface to facilitate the development of new measurement experiments and testing new ideas for predicting resource availability and optimizing measurement deployment.

## References

[1] Monsoon Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[2] Service: Process lifecycle. `https://developer.android.com/reference/android/app/Service.html#ProcessLifecycle`.

[3] University of Notre Dame NetSense Project: A study into the formation and evolution of social networks using mobile technology. `http://netsense.nd.edu/`.

[4] ALCATEL-LUCENT. 9900 wireless network guardian, 2013. http://www.alcatel-lucent.com/products/9900-wireless-network-guardian.

[5] AUYOUNG, A., BUONADONNA, P., CHUN, B. N., NG, C., PARKES, D. C., SHNEIDMAN, J., SNOEREN, A. C., AND VAHDAT, A. John Wiley & Sons, Inc., 2009, ch. Two Auction-Based Resource Allocation Environments: Design and Experience.

[6] CAPPOS, J., BESCHASTNIKH, I., KRISHNAMURTHY, A., AND ANDERSON, T. Seattle: A platform for educational cloud computing. In *SIGCSE* (2009).

[7] CARRIER COMPARE. Compare speed across networks. https://itunes.apple.com/us/app/carriercompare-compare-speed/id516075262?mt=8.

[8] CARRIER IQ. Carrier iq: What it is, what it isn't, and what you need to know. http://www.engadget.com/2011/12/01/carrier-iq-what-it-is-what-it-isnt-and-what-you-need-to/.

[9] CHALLEN, G. Phonelab testbed. http://www.phone-lab.org.

[10] CHOFFNES, D., BUSTAMANTE, F., AND GE, Z. Using the crowd to monitor the cloud: Network event detection from edge systems. In *Proc. of ACM SIGCOMM* (2010).

[11] CRANOR, C., JOHNSON, T., SPATASCHEK, O., AND SHKAPENYUK, V. Gigascope: a stream database for network applications. In *SIGMOD* (2003).

[12] F. QIAN, Z. WANG, A. GERBER, Z. M. MAO, S. SEN, AND O. SPATSCHECK. Characterizing Radio Resource Allocation for 3G Networks. In *Proc. of IMC* (2010).

[13] Fcc announces "measuring mobile america" program. http://www.fcc.gov/document/fcc-announces-measuring-mobile-america-program.

[14] GERBER, A., PANG, J., SPATSCHECK, O., AND VENKATARAMAN, S. Speed testing without speed tests: estimating achievable download speed from passive measurements. In *Proc. of IMC* (2010).

[15] HSIEH, C.-K., FALAKI, H., RAMANATHAN, N., TANGMUNARUNKIT, H., AND ESTRIN, D. Performance evaluation of android ipc for continuous sensing applications. *SIGMOBILE Mob. Comput. Commun. Rev.* (2012).

[16] HUANG, J., XU, Q., TIWANA, B., MAO, Z. M., ZHANG, M., AND BAHL, P. Anatomizing application performance differences on smartphones. In *Proc. of MobiSys* (2010).

[17] J. HUANG, F. QIAN, A. GERBER, Z. M. MAO, S. SEN, AND O. SPATSCHECK. A Close Examination of Perfomance and Power Characteristics of 4G LTE Networks. In *Proc. of MobiSys* (2012).

[18] JANG, K., HAN, M., CHO, S., RYU, H., LEE, J., LEE, Y., AND MOON, S. 3G and 3.5 G Wireless Network Performance Measured from Moving Cars and High-Speed Trains. In *in Proceedings of the 1st ACM workshop on Mobile internet through cellular networks* (2009).

[19] KORTUM, P., RAHMATI, A., SHEPARD, C., TOSSELL, C., AND ZHONG, L. LiveLab: Measuring wireless networks and smartphone users in the field. http://livelab.recg.rice.edu/traces.html.

[20] KREIBICH, C., WEAVER, N., NECHAEV, B., AND PAXSON, V. Netalyzr: Illuminating the edge network. In *IMC* (2010).

[21] LANER, M., SVOBODA, P., HASENLEITHNER, E., AND RUPP, M. Dissecting 3G Uplink Delay by Measuring in an Operational HSPA Network. In *Proc. PAM* (2011).

[22] Measurement Lab website. http://www.measurementlab.net/.

[23] Myspeedtest app. https://play.google.com/store/apps/details?id=com.num.

[24] NetRadar. https://www.netradar.org/en/about.

[25] NG, C., BUONADONNA, P., CHUN, B. N., SNOEREN, A. C., AND VAHDAT, A. Addressing strategic behavior in a deployed microeconomic resource allocator. In *Workshop on Economics of Peer-to-Peer Systems* (2005).

[26] OOKLA. Ookla speedtest mobile apps. http://www.speedtest.net/mobile/.

[27] P802.16.3 - standard for mobile broadband network performance measurements. http://standards.ieee.org/develop/project/802.16.3.html.

[28] PATRO, A., RAYANCHU, S., GRIEPENTROG, M., MA, Y., AND BANERJEE, S. The anatomy of a large mobile massively multiplayer online game. *MobiGames* (2012).

[29] PlanetLab website. http://www.planet-lab.org.

[30] PORTOLAN. The portolan network sensing architecture. http://portolan.iet.unipi.it/.

[31] QIAN, F., WANG, Z., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *Proc. of MobiSys* (2010).

[32] RIPE NCC. Ripe atlas. https://atlas.ripe.net/.

[33] ROMIRER-MAIERHOFER, P., RICCIATO, F., D'ALCONZO, A., FRANZAN, R., AND KARNER, W. Network-Wide Measurements of TCP RTT in 3G. In *Proceedings of the First International Workshop on Traffic Monitoring and Analysis* (2009), TMA.

[34] SCLAMP, J., AND CARLE, G. Measrdroid. http://media.net.in.tum.de/videoarchive/SS13/ilab2/2013+05+02_1000+MeasrDroid/pub/slides.pdf.

[35] SEN, S., YOON, J., HARE, J., ORMONT, J., AND BANERJEE, S. Can They Hear Me Now?: A Case for a Client-assisted Approach to Monitoring Wide-area Wireless Networks. In *Proc. ACM IMC* (2011).

[36] SOMMERS, J., AND BARFORD, P. Cell vs. WiFi: on the performance of metro area mobile connections. In *Proc. of IMC* (2012).

[37] Speedometer project source.

[38] Speedtest.net. http://www.speedtest.net/.

[39] Su, A.-J., Choffnes, D., Kuzmanovic, A., and Bustamante, F. Drafting behind Akamai: Travelocity-based detouring. In *Proc. of ACM SIGCOMM* (Pisa, Italy, September 2006).

[40] Vacirca, F., Ricciato, F., and Pilz, R. Large-Scale RTT Measurements from an Operational UMTS/GPRS Network. In *Proceedings of the First International Conference on Wireless Internet* (2005), WICON.

[41] Wang, Z., Qian, Z., Xu, Q., Mao, Z., and Zhang, M. An untold story of middleboxes in cellular networks. In *Proc. of ACM SIGCOMM* (2011).

[42] Xu, Q., Gerber, A., Mao, Z. M., and Pang, J. Acculoc: practical localization of performance measurements in 3g networks. In *Proc. of MobiSys* (2011).

[43] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M. Cellular data network infrastructure characterization and implication on mobile content placement. In *Proceedings of SIGMETRICS* (2011).

[44] Zarifis, K., Flach, T., Nori, S., Choffnes, D., Govindan, R., Katz-Bassett, E., Mao, Z. M., and Welsh, M. Diagnosing path inflation of mobile client traffic. Tech. Rep. 13-934, University of Southern California, 2013.