MobiScope: Pervasive Mobile Internet Traffic Monitoring Made Practical

Ashwin Rao
INRIA

Amy Tang
UC Berkeley

Adrian Sham
University of Washington

Sam Wilson
University of Washington

Shen Wang
University of Washington

Justine Sherry
UC Berkeley

Arnaud Legout
INRIA

Walid Dabbous
INRIA

Arvind Krishnamurthy
University of Washington

David Choffnes
University of Washington

ABSTRACT

Characterizing Internet traffic naturally generated by mobile devices remains an open problem. The key challenge is that mobile devices and their OSes provide no built-in service for monitoring and reporting all network traffic. The result is that researchers are left with partial views of network activity– through monitoring inside mobile carrier networks, from WiFi access points or logging data on custom OSes.

In this paper, we take an alternative approach: measurement through indirection. Specifically, we exploit the fact that most mobile OSes support proxying via virtual private networks (VPNs). By sending mobile Internet traffic through a proxy server under our control, we can monitor all flows regardless of device, OS or access technology. Further, our solution is amenable to large-scale deployment because it requires no special privileges and can be configured via software on users' existing phones.

We report the results of a 6-month IRB-approved measurement study using this approach both in the lab environment and with human subjects in the wild. After demonstrating that our approach incurs reasonable overheads, we describe our measurement methodology and how we use *MobiScope* to measure the impact of device OS, apps and service provider on Internet traffic.

[**TBD: Monitoring?– We also perform controlled experiments**]

1. INTRODUCTION

Mobile systems consist of walled gardens inside gated communities, *i.e.*, locked-down operating systems running on devices that interact over a closed and opaque mobile network. As a result, characterizing Internet traffic naturally generated by mobile devices remains an open problem. [**TBD: Why do we care?**]

The key challenge is that mobile devices and their OSes provide no built-in service for monitoring and reporting all network traffic. Previous work has provided us with only partial views of network activity. For example, researchers with privileged access to data gathered from a cellular carrier's network can report on activity for a large group of users; however, this view does not cover WiFi traffic nor does it speak to traffic generated through other cellular networks. [**?**] Likewise, studies that instrument an enterprise WiFi network can view traffic from devices subscribing to multiple cellular carriers, but not the traffic flowing over cellular links [**?**]. Last, researchers have used a fleet of devices running a custom version of Android to perform detailed logging of network activity [**?**]. This can capture all of a device's network traffic but is restricted to measuring Android behavior. Further, subjects must root their own devices or to switch from their primary device for the duration of the study – barriers that can limit deployment size and introduce bias. [**TBD: JS: Concerned about all of this related work – seems like there's a lot! Also – why is the fact that WiFi vs Celular traffic a big deal? Apps are apps right? Data is data? I guess performance questions come up but isn't okay that the two are characterized separately? Further, meddle introduces big performance changes so...**]

In this paper, we take an alternative approach: measurement through indirection. Specifically, we exploit the fact that most mobile OSes support proxying via virtual private networks (VPNs). By sending mobile Internet traffic through a proxy server under our control (an approach we call *MobiScope*), we can monitor all flows regardless of device, OS or access technology. Importantly, installing a VPN configuration requires neither a new app to be installed nor does it require special or new privileges, thus facilitating large-scale deployment on unmodified device OSes.

We report the results of a 6-month IRB-approved measurement study using this approach both in the lab environment and with human subjects in the wild. After demonstrating that our approach incurs reasonable overheads, we describe our measurement methodology and how we use *Mo-*

biScope to measure the impact of device OS, apps and service provider on Internet traffic.

Our key contributions are as follows:

- We demonstrate the feasibility of proxy-based measurement for characterizing mobile Internet traffic for iOS and Android. *MobiScope* captures all Internet traffic generally with less than 10% power and packet overheads, and negligible additional latency. We will make the *MobiScope* software and configuration details publicly available. [**TBD: JS: what does "we will make this publicly available" mean? You mean that, for anonymous submission, details have been obscured but on publication we will make this all public? Confused.**]
- A descriptive analysis of network traffic naturally generated by devices in the wild, across different access technologies. We find, for example, that mobile traffic volumes are approximately equally split across WiFi and cellular – highlighting the importance of capturing both interfaces. Further, we find that most traffic is compressed, limiting the opportunities for additional traffic-volume optimization.
- We characterize the network traffic generated by mobile OSes, and how it varies when using different access technologies.
- A measurement study of app behavior (both popular and otherwise) from Android and iOS. We observe [**TBD: values come here**]. [**TBD: say something about how we can directly observe differences in the network behavior of identical apps designed for different OSes.**]
- An analysis of privacy leaks in the mobile environment. [**TBD: Results based on Amy work**].
- A new measurement technique for detecting ISP interference with arbitrary Web site content.
- [**TBD: Results from an on going IRB based study of 30 users. We use these results to compare our observations from exisiting studies. The key take home is that these measurements were did not require custom OSes, ISP support, or support from marketplaces, warranty voiding of devices.**]

[**TBD: The above is a laundry list – can we highlight three or four things at most? Sort of macro points and get to the details later?**]

The remainder of the paper is organized as follows:

[**TBD: Things to highlight in Intro
Tools
Techniques
Methodology
Insights**]

[**TBD: Justine: Primary concern is that the secondary paragraph doesn't sell this as very novel – others have all done this before is sort of the lesson I learned there. What's new?**

After reading this, I think we need to say, "comprehensive network usage analysis" is part of what's new here - we can track users across multiple networks and platforms; this allows us to say that x fraction of traffic is over 3G and y fraction is over WiFi, that bandwidth usage changes by x percent wen moving betwen 3G and Wifi." Because it's easy to install, this means that we can study large numbers of people (given IRB constraints) with little overhead.

One additional thing is we should call out what findings we have are new – it doesn't matter if our methodology is new at all if we have sexy new discovery X property of network traffic/app behavior/etc.]

2. BACKGROUND

[Dave: This is sending the wrong message. We want to say there is a set of trade-offs when measuring network traffic from mobile devices. Previous work compromised network coverage, portability, deployability and/or overhead. In this work, we present an approach that compromises none of these, potentially enabling a large-scale deployment across carriers, devices and access technologies. Our prototype measurement system, as part of an IRB-approved study, shows the potential of this approach.]

[Dave: Other thing to emphasize, which is closer to what is in the following paragraphs: What are researchers using the data for? Security, privacy, traffic characterization for power/cost.]

Despite the increasing popularity of mobile devices, the current mobile ecosystem offers researchers a limited view into mobile Internet traffic. Intuitively, the factors that affect mobile Internet traffic include mobile OS and application design decisions, the access technologies used to connect to the Internet, and the ISP policies. This diversity in factors implies that characterizing mobile Internet traffic requires end user participation in measurement studies and controlled experiments. Current techniques to characterize mobile Internet traffic include instrumenting the mobile operating system (OS), instrumenting application binaries, static analysis of application binaries, and relying on ISP traces. We now show that these techniques are not practical for end user participation.

Instrumenting a mobile OS system using tools such as Taintdroid [2] and AppFence [3] provides researchers a fine grained view of the apps and OS in action. However, this fine grained view comes at a high cost of jail-breaking and warranty voiding of the devices. Most end users shall not be willing to pay this cost. Instrumenting an mobile OS is therefore not practical for measurement studies and experiments that require end user participation. Furthermore, longitudinal studies that detail the impact of OS code changes and application code changes cannot be performed by instrumenting OSes.

	Network Coverage	Portability	Deployment model	Meas. Type
AT&T study	Single carrier	All OSes	Instrument cell infrastructure	Passive
WiFi study	Single WiFi network	All OSes	Instrument WiFi network	Passive
MobileLab	Multiple networks	Android	Install Rooted OS	Active/Passive
MobiPerf	Multiple networks	Android	Install App	Active
MobiScope	Any network	Android / iOS	VPN configuration	Passive

Table 1: Comparison of alternative measurement approaches. *MobiScope* is the first approach to cover all access networks and most device OSes, capturing network traffic passively and with low overhead via VPN proxying.

Instrumenting app binaries using tools such as AppInsight [5] can be used to characterize the network traffic from mobile applications. Indeed, AppInsight provides a detail analysis of mobile applications. However, in terms of the network footprint of the app, the scope of AppInsight is limited to the instrumented apps, the marketplaces from where the apps are downloaded, and the OS version for which the app was instrumented. Furthermore, each new version of the app needs to be instrumented to characterize the impact of the changes.

Static analysis of the app code can be used to study mobile application whose code cannot be instrumented. For example, PiOS [1] was used to perform static analysis of 1400 IOS apps. A shortcoming of PiOS is that access to the app binary is possible only if the device is jail-broken, thus voiding the warranty of the device. Furthermore, like AppInsight [5], the results of PiOS are limited to the iOS operating system. [**TBD: Dave Text for SPARTA project at UW.**]

ISP traces are useful to study mobile devices in the wild. Vallina-Rodriguez *et al.* [8] use an ISP trace of 3 million subscribers to detail the impact of ads and analytics on mobile Internet traffic and energy consumption. Similarly, Maier *et al.* [4] study the mobile traffic by looking at the DSL traces from a popular European ISP. However, the data used in these studies is limited to the ISP that provided these traces. Mobile devices can use different ISPs depending on their location and the access technology used to connect to the Internet. For example, the home Wi-Fi and office Wi-Fi may be served from ISPs that are different from the ISP used for cellular data traffic. Therefore, traces from a single ISP are expected to have a limited view on the traffic from the mobile devices.

ISPs can interfere with the Internet traffic to inject javascript code for advertisements and analytics. This problem of ISP interference was highlighted by Reis *et al.* [6]. The authors demonstrated that inflight changes made by ISPs tend to introduce vulnerabilities such as overflows and cross-site scripting (XSS) attacks. They proposed and deployed *Web Tripwires*, a Javascript code that detects in-flight page changes. The main limitation of *Web Tripwires* is that Web sites are required to modify their content to include a tripwire that can detect ISP interference. *Web Tripwires* are therefore not practical because they require support from the Web site maintainers. [Dave: does not belong here]

In summary, existing solutions to measure the network characteristics of mobile Internet traffic fall short of being either portable, pervasive, or ISP agnostic. Furthermore, the need for practical monitoring on mobile Internet traffic becomes more critical with the possible arrival of new mobile operating systems from the Ubuntu and Firefox communities. In the next section we show how traffic redirection can be to monitor mobile Internet traffic.

3. MOBISCOPE OVERVIEW

[**TBD: Should we explicitly mention traffic redirection as a tool?**] This section describes the goals for our monitoring approach, and how *MobiScope* achieves these goals using proxying. We show that our approach imposes reasonable overheads, and we describe how our IRB-approved study protects user privacy.

3.1 Goals

Our primary goal is *to monitor all the Internet traffic from mobile devices regardless of the operating system, wireless access technology, and the ISPs used by the mobile device*. To achieve this goal, we identify the following desirable properties for a measurement platform:

1. *Portable.* Our approach should work on all major device OSes without requiring support from carriers or ISPs.
2. *Pervasive.* We should be able to measure traffic regardless of the location, access technology, and ISPs used by mobile devices.
3. *Passive.* We wish to understand the network traffic naturally generated by users and their devices, requiring passive monitoring.
4. *Deployable.* We want a low barrier to entry to facilitate large-scale adoption with minimal impact on the user experience.

[**TBD: These points need to be revisited to incorporate Trip wires.**]

3.2 Approach

We now describe how we achieve the goals in the previous section using proxying. Specifically, we support two approaches to proxying mobile traffic: secure proxying via virtual private networks (VPNs) and insecure transparent proxying. These two approaches allow us to measure traffic with and without carrier interposition, respectively. Because the VPNs are natively supported by most device OSes, we focus our analysis on results gathered via this approach.

3.2.1 VPN Proxying

The key observation that enables our approach is that most device OSes support VPNs out of the box, in large part to meet the security and connectivity goals of enterprise customers. Instead of using a VPN server as a gateway to a private network, we slightly abuse the feature to proxy Internet traffic. Android, BlackBerry, Bada, and iOS all support VPNs tunnels, and those tunnels are effective both over Wi-Fi and the cellular interface.

Broad support for VPN connectivity meets our goals for portable, passive and deployable monitoring of mobile Internet traffic, but to meet our goal of pervasiveness we must ensure that the VPN proxy is *always* enabled. Currently, all iOS devices (version 3.0 and above) support this using feature called "VPN On-Demand". *VPN On-Demand* forces the iOS device to use VPN tunnels when connecting to a specified set of domains. Using trial-and-error, we discovered that VPN On-Demand uses suffix matching to determine which domains require a VPN connection. We use this knowledge to configure *VPN On-Demand* such that iOS devices use a VPN tunnel to access the Internet. For Android devices, version 4.2 and above supports "Always On" VPN connections that are established regardless of the destination for network traffic. For Android version 4.0 and above, we support this functionality via an app that uses only standard Android APIs.

Both Android and iOS support IPSec for establishing VPN tunnels. *MobiScope* uses Strongswan [7] as the VPN server because it is the only open source solution that uses native IPSec in Linux *without any kernel modifications*.

3.2.2 Insecure Transparent Proxy

VPN proxying via secure tunnels prevent ISPs from inspecting or interposing on network traffic, thus preventing us from measuring this behavior. To understand ISP interference with mobile Internet traffic, we additionally support measurement using an *insecure* transparent proxy. In particular, we exploit Android's support for apps providing VPN services; instead of establishing a secure connection we simply forward traffic to a proxy server without additional encryption. In this way, the mobile device's ISP can inspect the contents of all non-SSL traffic and interpose accordingly. Note that one limitation of this approach is that the ISP will see the destination for all network traffic is our proxy server (instead of the original destination), which could impact how ISPs treat the corresponding traffic.

3.2.3 Traffic Monitoring and Ethics

As shown in Figure 9, mobile devices proxy their Internet traffic through a *MobiScope* server that monitors all their Internet traffic. We use *tcpdump* to monitor the traffic that passes through *MobiScope* servers. We will make the configurations and code for *MobiScope* public and open source.

Capturing all of a subject's Internet traffic raises significant privacy concerns. Our IRB-approved study entails informed consent from subjects who are interviewed in lab, where the risks and benefits of our study are clearly explained. Subjects are incentivized to use the VPN though a lottery for Amazon.com gift certificates. All data from tcpdump is encrypted before touching persistent storage; the private key is maintained on separate secure severs and only approved researchers can access it. Users may delete their data and/or disable monitoring at any time. For privacy reasons, we cannot make this data publicly available.

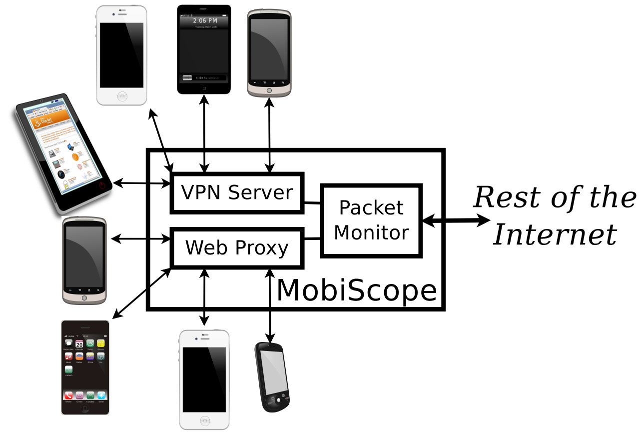

Figure 1: *MobiScope* uses traffic proxying to monitor mobile Internet traffic. *MobiScope requires mobile clients to redirect their traffic through a server that monitors Internet traffic. VPN based proxying is used to characterize mobile traffic sans ISP interference. To detect ISP interference* MobiScope *relies on a single hop transparent Web Proxy.*

[**TBD: Dave: New figure for Tripnet comes here.**]

Figure 2: Overview of the Web tripnet

[**TBD: Dave: Figure 2 text for tripnet should come here**]

3.3 Feasibility

We now show that the cost to proxy traffic through *MobiScope* is reasonably low in terms of latency, data consumption, and power.

3.3.1 VPN Latency Overheads

The iOS devices use IKEv1 to manage the VPN tunnels while Android devices support both IKEv1 and IKEv2. To establish the VPN tunnel, IKEv1 requires a total of 16 packets to be exchanged between the mobile client and the VPN server while IKEv2 requires 4 packets. *MobiScope* uses IKEv2 for Android devices while IKEv1 is used for iOS devices.

To measure the time required to establish a VPN tunnel, we performed controlled tests using one Android device and an iPhone 5. We performed these tests from two different locations based in the same city in which the server was deployed. OUr tests involved **!number**$_{verify}$! of VPN tunnel creation over a time period of $!_{verify}!$ hours. When the Android device used Wi-Fi to establish the VPN tunnel, we observe a median connection establishment time of 0.62 seconds from both locations with a maximum of 0.81 seconds and 0.79 seconds respectively. When the Android device used cellular networks to establish the tunnel, the median connection establishment time was 0.81 seconds from both locations with a maximum of 1.59 seconds. Compared to the Android device, the iOS devices required a larger amount of time to establish the connection because it relies on a an older key authentication protocol. From the two Wi-Fi networks, to establish the VPN tunnel, the iOS device required 1.60 seconds and 1.34 seconds with a maximum of 2.0 seconds and 1.48 seconds respectively; in the case of cellular networks we observed a median of 1.80 seconds and 1.65 seconds with a maximum of 2.18 seconds and 1.87 seconds respectively. [Dave: This needs to go in a table, since it is impossibly dense.]

[**TBD: In summary, we observe that because iOS devices use an older key exchange protocol they can take up to twice as much time as Android devices to establish the VPN tunnel. Any more insights .. The tunnel establishment times in the order of 2 seconds implies that *MobiScope* can have a significant latency overhead if VPN tunnels are established periodically for short tests.**]

3.3.2 Data Consumption Increase when using VPN

IPSec encapsulation slightly inflates packet sizes, in addition to preventing carrier middleboxes from applying their own compression. We measured the overhead of the tunnel in terms of data overhead from IPsec headers and keep-alive messages, finding that it ranges from 8–12.8%.

To to compute the increase in the amount of bytes transferred due to encapsulation and the keep-alive messages, we log the packet lengths of the encrypted packets (IPsec packets) exchanged by our *MobiScope* servers and the mobile clients. We performed this packet capture for 30 days during which 25 devices tunneled their traffic via *MobiScope*. During this time interval we also log the packet length of the packets encapsulated within the IPsec packets. During this 30 day period we observe that the median of the increase to be 8.31%, with a maximum increase of 12.8%.

[**TBD: In summary, we observe a maximum overhead of 12.8% increase in data consumption. We believe the costs of this overhead are minimal compared to the cost of warrant voiding the device.**]

3.3.3 Power Overheads when using VPNs

We found that VPN proxying imposes approximately 10% power overhead compared to direct traffic. To test the additional power consumption from using a VPN proxy, we used a power meter to measure the draw from a Galaxy Nexus running Android 4.2.[1] While instrumented, we conducted 10-minute experiments where we scripted device usage with and without a VPN enabled. The activities included Web searches, map searches, Facebook interaction, e-mail and video streaming.

3.4 Caveats

Proxying mobile Internet traffic provides a low-cost, portable, pervasive and deployable solution for measurement; however, there are several caveats for our approach.

1. When device traffic is proxied, Web sites that use client IP addresses to offer custom services will respond according the IP address of the *MobiScope* server.
2. Some ISPs are known to block VPNs. During our measurement we observed one such ISP that blocked VPN tunnel creation requests from one of our clients.
3. We observe that mobile devices currently create a VPN on at most one wireless interface. This implies that *MobiScope* cannot monitor traffic when the mobile device simultaneously uses more than one wireless interface. [Dave: Does this ever happen?]
4. We performed controlled experiments to test IPv6 support. We observe that though iOS and Android support IPv6 they currently do not support IPv6 traffic through VPN tunnels.
5. When using VPNs, *MobiScope* cannot monitor the traffic between the mobile device and access point used to connect to the Internet. [Dave: This is unclear to me.]

[**TBD: In summary, despite these shortcoming we believe that *MobiScope* can be used for realistic measurements of mobile Internet traffic.**]

4. METHODOLOGY AND DATASET

[**TBD: A bit of concern about separating out the methodology and results section – because there is so much data and so many different experiments, I worry that reviewers when readin the results will forget how we derived those results. And vice versa – that the reviewer, whilst reading the methodology, will forget why they should care about the methods described. Wouldn't it be better to describe the results bit by bit, and explain the methodology in-line, as needed? –Justine**]

In this section, we detail the data collection methodology of *MobiScope* and the datasets we collected. Our objectives were both to characterize the Internet usage properties of both specific applications and platforms, as well as to more generally characterize typical Internet usage properties given user behavior. Consequently, our measurements come in two categories: first, a set of controlled, deliberate experiments to study the properties of specific apps or platforms; and second, an N-month long 'in the wild study' of traffic generated by real Internet users who ran the *MobiScope* software on their personal smartphones.

[1] We use Android because power measurements require physical access to the battery for a device, which is not feasible for iOS devices. We found similar results when testing the time to discharge an iOS device while streaming video with and without a VPN connection.

4.1 Controlled Experiments

In our controlled experiments, we installed selected applications from Google Play/ theiPhone App Store on 'clean slate' Android/iOS devices with the latest versions of their operating systems (Android Jellybean 4.2 and iOS 5 respectively). After installing the app, we engaged in controlled behavior – detailed below – with the app while running the *MobiScope* software. After several minutes of interaction, we uninstalled the application and installed a new app. Our controlled experiments allowed us to study (1) bandwidth and battery impacts of iOS push notifications; (2) traffic patterns and usage for Android and iOS apps; and (3) leakage of personally identifiable information (PII) due to Android and iOS apps.

iOS Push Notifications. The applications running on iOS devices receive notifications from Web services using *iOS push notification*. Push notifications allow an application to alert the user of updates (*e.g.*, Facebook messages) while the phone is idle/not in active use. iPhone users are typically warned not to enable push notifications for too many apps due to the potential for these background tasks to (1) consume bandwidth resources and (2) consume battery resources, all without any active user behavior. However, these warnings come with little quantification of exactly *how much* an application's push notifications might impact battery life or bandwidth; to date the research community has not measured these properties due to the closed-source nature of iOS and consequent difficulty to measure these properties. Nevertheless, with *MobiScope* we can monitor the traffic generated due to push notifications and thus quantify the impact of push notifications despite the iOS lockdown; to the best of our knowledge this is the first measurement characterization of iOS push notifications.

Our experiments to study push notifications proceeded as follows:[**TBD: ...Ashwin?**]

The results of these tests can be found in §**??**.

Android Applications. Both Android and iPhone apps generate traffic to load and upload user data, app content, and advertisements. Although users are informed upon application installation whether or not an app is allowed to access the Internet, the user is unaware *what* data is sent, *how much* data is sent or accessed, or *with whom* the app communications. We define a 'well-behaved' application as one which (a) makes limited use of network and battery resources (*i.e.* by accessing little bandwidth and by batching traffic to allow radio shutdown during idle periods); (b) contacts only those servers necessary to perform application behavior (*i.e.* contacting only a limited number of advertising networks and no tracking sites); and (c) not leaking any personally identifiable information over the network, (*i.e.* using HTTPS whenever uploading needed private information like email addresses, and never uploading unnecessary personal information like address book contents or device IMEI). [**TBD: Justine: I need your help to rewrite the text here. I have put some crappy text as placeholder.**]

We test how many applications actually meet these criteria of well-behaved network usage, we performed controlled experiments on blank Android smartphones, iteratively installing, playing with, and monitoring the behavior of hundreds of Android apps whilst running the *MobiScope* app in the background. We tested the top 100 most popular free Android apps manually – installing each app by hand, entering user credentials for accounts like Facebook and Twitter, and toying with the app. In addition to this manual setup, we used an automatic test-click generator to further toy with the app. Afterwards, we uninstalled the app and reset the device.

Android, unlike iOS, allows users to 'side-load' third-party apps on to their device; consequently there are numerous third-party app markets on the web in addition to Google's official Play Store. To study these apps, we performed fully-automated tests on 1003 apps from a free, third-party app market. Our automation used the adb Android command shell to install each app, enable *MobiScope*, and start the app. The system then used Monkey, the built-in adb stress tool, to perform a series of 10,000 actions. These actions consisted of random swipes, touches, and text entries. The system then once again used adb to uninstall the app and reboot the device (thus ending all lingering connections and metadata from the previous app.)

The results of these tests can be found in §**??**.

iPhone Applications. [**TBD: Dave/Ashwin: Complete the text here for iOS – subset of apps**]. Dummy text for the paragraph. [**TBD: Dave: Text Here**]

4.2 In The Wild

Along with controlled experiments we also conducted a measurement study to characterize the mobile Internet in the wild. We deployed two *MobiScope* servers in USA and one server in France. These servers tunnel Internet traffic using VPNs from 25 devices, belonging to 19 users who are volunteers for our IRB approved study. To protect the identity of the users and their data, on each server we use public key cryptography to encrypt the files that log the data traffic that flow through the server. We call this dataset the *mobAll* dataset.

The 25 devices that contribute to the *mobAll* dataset consists of 10 iPhones, 4 iPads, 1 iPodTouch, 9 Android phones, and 1 Android tablet. Though *tablets* can access the Internet via a cellular data connections, for the *mobAll* we included tablets that only use Wi-Fi to access the Internet. The Android devices in this dataset include the Nexus, Sony, Samsung, and Gsmart brands.

This dataset consists of 202 days of data that flowed through our VPN servers; the number days for each user varies from 5 to 198 with a median of 35 days.

We estimate the access technology used by the mobile device by performing a *WHOIS* lookup on the IP address used by the mobile client for creation of the VPN tunnel. We use the WHOIS databases available at *whois.cmyru.com* and *utrace.de* to get the ISP details. We observe that ISPs that provide Internet access over cellular connections use dedicated ASes for cellular traffic. We use the information provided by the *WHOIS* databases to manually classify the ASes used by the mobile devices to be either cellular or Wi-Fi. This classification gives incorrect results when mobile clients are served by a Wi-Fi access point that internally uses a cellular connection to connect the Internet. In this case, though the device uses Wi-Fi to connect to the Internet, our servers will log the connection to be from a cellular ISP.

[**TBD: we need some wording and consitency for the usage of ISP – for example ATT can provide cellular and DSL. Also mobile data cannot be used and we need some word for cellular data and wifi data and this must be defined in the dataset description.**]

Based on the above classification of access technology and ISPs, our dataset consists of data traffic from 52 distinct ISPs, of which 10 provided cellular services. Of the 18 devices that used cellular data, we observed that 15 devices restricted their cellular data traffic to one ISP each; we observed that the other three devices accessed the Internet using the services of two different ISPs. We observed that the devices in our dataset used a higher number of Wi-Fi ISPs. We observed a median of 4 Wi-Fi ISPs per device with a maximum of 25 Wi-Fi ISPs that were used by one device. This observation confirms our intuition that studies based traces from a single ISP [4, 8], shall not be able to analyze how specific users use mobile devices.

4.3 Discussion

[**TBD: In summary, ...**]

5. NETWORK CHARACTERISTICS OF OS SERVICES

Mobile operating systems provide APIs and OS level services to optimize network usage. For example, the Apple Push Notification service (APNs) and Google Cloud Messaging (GCM) are used by iOS applications and Android applications respectively to receive notifications from the Internet. [**TBD: About location services**] In this section we perform a set of controlled experiments to detail the network characteristics of these OS services. The questions that we answer in this section are as follows.

1. What are the network characteristics of operating system services?
2. How different is the network traffic from iOS devices compared to Android devices?
3. What is the impact of operating system services in the wild? [**TBD: rephrase this**]

5.1 Traffic from Factory Reset Devices

We now detail the network characteristics of mobile devices and the pre-installed applications. We use the following questions to guide our analysis.

1. What is the network usage of devices that are used *out of the box*?
2. How does the device, manufacturer, and operating system affect the network usage?

Application	**Traffic Share in the first 24 hours**			
	iPad (19 KB)	iPod (21 KB)	Galaxy SIII (47 KB)	Nexus (97 KB)
Notifications	0.54	0.53	0.35	0.88
Location	0	0	0.26	0
SSL	0	0	0.30	0.11
Mail	0.05	0.07	0	0
HTTP	0.13	0	0.09	0
UDP	0.28	0.40	0.01	0.01
total	*1.0*	*1.0*	*1.0*	*1.0*

Table 2: Network usage in the first 24 hours after factory reset. *Notifications contribute to the largest fraction of traffic volume across all devices.*

We answer these questions with a controlled experiment performed on an iPod Touch, an iPad, a Samsung Galaxy SIII, and a Google Nexus S Phone. Each of these devices were reset to factory default settings after their batteries were fully charged. We then allowed these devices to connect to the Internet using our Wi-Fi hotspot. We ran tcpdump on our hotspot to monitors the Internet traffic from these devices for 3 sessions of 24 hours. We add a dummy email account as the primary account on each of these devices. This account is responsible for triggering any OS specific services that may require the device to be in use. We use the data collected as a rough estimate on the minimum data traffic that is generated by the devices. We observed that during the initialization the devices exchanged from 20 MB to 50 MB. We now present the traffic characteristic observed during the time after this initialization.

In Table 2, we present the observed the traffic share of OS notification services and other services. We use the IP protocol and TCP port numbers to classify these services: TCP port 80 as HTTP, TCP port 223 as SSL, TCP port 993 as Mail, UDP flows as UDP, and so on. For the iOS devices, in Table 2 we observe that notifications contribute to 54% of the traffic volume. For the Android devices, we observe that the Nexus phone receives far more notifications (88% of 97 KB) compared to the Samsung Galaxy SIII phone (35% of 47KB). We believe that this difference is because the Android phones came with a different set of pre-installed applications depending on their vendor. Furthermore, we observe that in the first 24 hours the Samsung Galaxy SIII phone used the open mobile alliance location protocol; we did not observe the usage of this protocol by the Google Nexus S phone in the first 24 hours. All the UDP flows were DNS requests.

The push notifications messages, that contributed to the maximum traffic share in Table 2, were exchanged over TCP. The iOS devices used TCP port 5223 while the Android devices used TCP port 5228 for the push notifications. The notification services require a TCP connection to be established by the device and the server. The notification server uses this TCP connection to push notifications to the mobile

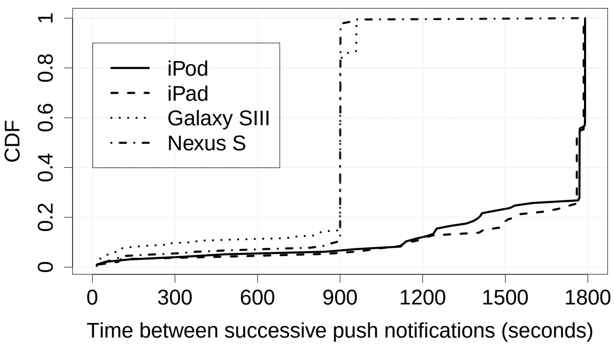

Figure 3: Inter-arrival time between push notification messages after factory reset. *The iOS devices communicate with the notification server approximately once every 1800 seconds while the Android devices communicate once every 900 seconds with their notification server. [***TBD: Add results from shen from iphone 3gs reset.***]*

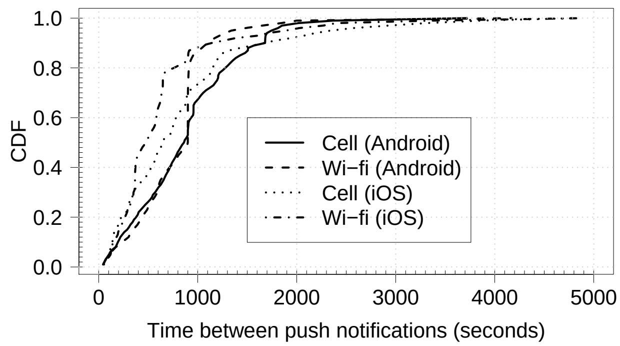

Figure 4: Distribution of the time between push notification messages in the wild. *The frequency of push notification messages is higher for the iOS devices in our dataset compared to the Android devices. Notification messages are less frequent over cellular networks compared to Wi-Fi networks.*

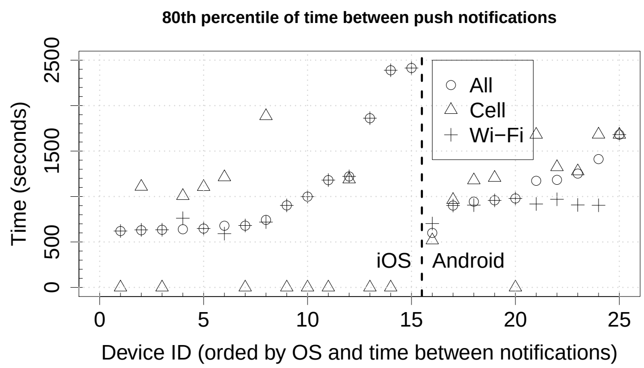

Figure 5: Inter-arrival time between push notifications in the wild. *Push notifications occur less frequently over Cellular networks. The rate of push notifications depends on users and the applications installed. [***TBD: DISCUSS: Better representation for tablets – currently they have a value of 0 for the cell networks.***]*

device. The mobile device also periodically communicates with the notification server.

In Figure 3 we plot the time between successive messages exchanged on the ports used for push notifications. We observe that the inter-arrival time between push notifications for the Android devices is 900 seconds for more than 80% of the push notifications observed. For the iOS devices, we observe an inter-arrival time at least 1700 seconds for that more than 75% of the push notifications. All Android flows with an inter-arrival time larger than 800 seconds consisted of an empty TCP packet sent by the device followed by a 25 byte payload sent by the server. All iOS flows with an inter-arrival time larger than 1500 seconds began with an TCP packet with a payload of 85 bytes sent by the device followed by the server responding with of a TCP packet of 37 byte payload.

5.2 Push Notifications In The Wild

We now characterize our observations on the push notifications we observed in the *mobAll* dataset. The objective of this analysis was to answer the following questions

1. How frequently do Push notifications take place in the wild?
2. What is the impact of access technology on push notifications?
3. [**TBD: What is the distribution of traffic volume of push notifications?**]
4. [**TBD: How do push notifications change over OS and device upgrades?**]
5. [**TBD: Do not disturb – How efficient are services like Do Not Disturb?**]

In Figure 4 we plot the distribution of the time between successive push notification messages for Android and iOS devices over cellular and Wi-Fi networks. While computing this distribution, we account the diversity in device usage in the following manner. For each device and each access technology we compute the 100 quantiles from 0.01 to 1.0 in steps of 0.01 of the time between successive push notifications. We then use the median value of each quantile (from 0.01 to 1.0 in steps of 0.01) for a given access technology and operating system of the device. In 3 we observe a higher time between push notifications on cellular networks compared to Wi-Fi networks. We also observe that the time between push notifications is higher for the iOS devices in our dataset compared to the Android devices in our dataset. [**TBD: The tcp ports used after the push notifications. Numbers for what fraction was ssl traffic at 443. and the servers to which the connection was made.**]

In Figure 5 we present the time between successive push notifications for the 25 devices in our dataset. As observed in Figure 4 we observe that the iOS devices receive push

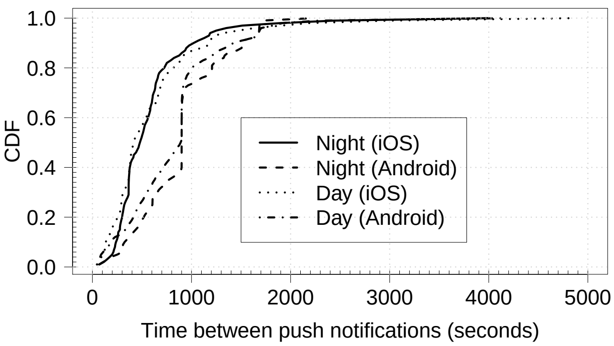

Figure 6: Impact of time-of-day on the push notifications. *The rate of push notifications is agnostic of the time of the day for iOS devices.*

messages more frequently that the Android devices. We also observe that the time between push notifications is higher for Android devices. The iOS devices prefer a cellular data connection for Push notification over Wi-Fi [**TBD: http://support.apple.com/kb/TS4264**]. However, in Figure 4 and Figure 4 despite this preference, we observe that the time between successive push notifications for iOS devices is higher over cellular networks in comparison to Wi-Fi networks. We observe that [**TBD: SSL traffic**] to mail servers was followed [**TBD: x%**] after push notifications. This implies that higher usage of the device over Wi-Fi may result in a higher number of notificatons received. In Figure 5, device ID

[**TBD: Highlight the pervasive nature of *MobiScope* allows us**] We now use Figure 6 we to show that the push notification are agnostic of the time of the day. For Figure 6, we consider two time periods: from midnight to 6 am (*Night*) and from 6 am to midnight (*Day*). The values used in the distributions is computed using the technique used for 4. We observe that the Android and iOS devices exhibit a similar behavior that appears to be agnostic of the time of the day. The iOS devices (from verion 6.0) come with a feature called *Do Not Disturb (DND)* that does raise notification alarms on receiving notifications during specific time periods. We also observe that during the intervals configured as *Do Not Disturb*, notification messages were exchanged by devices that used this feature enabled.

[**TBD: Who is pushing the notifications. Servers used for push notifications..based on DNS requests responses**]

5.3 Location Services In The Wild

5.4 Discussion

6. APPLICATION CHARACTERIZATION

We now turn to measurements of specific popular iOS and Android applications. When users install apps, they grant them Internet access without detailed knowledge of how that access will be used, including *how much* data is sent or accessed, *what* data is sent, or *with whom* the app communications. "How much" is important to conserve both bandwidth caps and battery capacity: an app which consumes or produces too much data will waste bandwidth resources, while an app which consumes or produces data too frequently will prevent the device radio from going idle to save power. "With whom" is important to protect users from excessive tracking – the more organization's servers an app connects to, the more organizations which are able to track user behavior, location, or other private data. Finally, "what data" is important because apps may unnecessarily leak personally identifiable information (PII) such as user email address, IMEI, contact information, or other stored data either to the app provider or worse, to any eavesdropper on a public WiFi connection. We report on our findings in all three of these dimensions for the iPhone and Android apps in our study.

6.1 Bandwidth and Radio Usage

In the Wild.

- Stats on how much bandwidth each user used; time of day; how frequent...

Android Apps. To dig in to the root cause of these usage patterns, we also did an 'app-by-app' analysis of network usage to see if most bandwidth consumption/radio time was the result of a few heavy applications, with most applications relatively idle, or whether usage was divided amongst all applications equally. In Figure **??**, we plot the CDF of total bytes transferred by each app in our study, one line for the top-100 Google Play apps we tested manually, and another for the top 2000 apps, tested automatically, from a third-party market. We see that...[**TBD: Amy...**] Regarding radio usage,...[**TBD: Do we even have time to do this? I don't remember the exact metrics we used for the MobiSys submission.**]

iPhone Apps.

6.2 Third Party Servers

Many free applications support themselves financially by serving ads or providing resources for third parties to track user behavior. We now explore how many servers are contacted by a given app (*i.e.* how many providers are tracking a user with this app) – most of these typically for ads, tracking, or analytics – as well as how much data is transferred to and from these servers (*i.e.* how much does this traffic impact the user's data cap?).

In the Wild. We first consider the overall impact of these ads, analytic, and tracking services on typical user behavior in our IRB study... [**TBD: Ashwin...**]

Android Apps. When we inspect the data from our controlled study, we see that some apps contact a large number of external servers while others contact significantly fewer. In Figure **??**, we show both the total number of servers contacted (solid lines) as well as the number of organizations contacted (dotted lines) for both the top-100 Google Play dataset and the top-2000 third-party dataset. To quantify

Dataset	Platform	# Apps	Email	Location	Name	Password	Android ID	Contacts	IMEI
Google Play	Android	100	3 (3%)	10 (10%)	2 (2%)	1 (1%)	21 (21%)	0 (0%)	13 (13%)
Third Party	Android	908	1 (0.1%)	32 (3.5%)	2 (0.2%)	0 (0%)	95 (10.4%)	4 (0.4%)	48 (5.3%)
App Store	iPhone	100	?	?	?	?	?	?	?

Table 4: Summary of personally identifiable information leaked in plaintext (HTTP) by Android and iPhone apps.

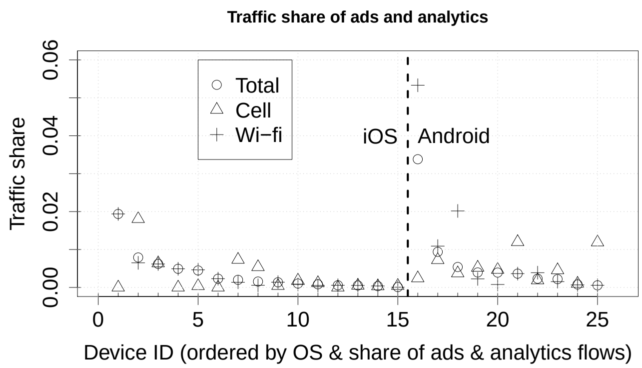

Figure 7: Fraction of traffic volume because of Ads and Analytics. *[**TBD: Check for id1 and id25**]*

Tracker	Number of devices tracked		
	Total	**Android**	**iOS**
doubleclick.net	25	11	14
google-analytics.com	25	11	14
googlesyndication.com	22	10	12
admob.com	21	10	11
scorecardresearch.com	21	10	11
2mdn.net	20	9	11
atdmt.com	18	9	9
imrworldwide.com	18	9	9
flurry.com	17	7	10
googleadservices.com	17	8	9

Table 3: The top 10 ads and analytics sites that tracked the devices in our dataset. *Two trackers,* doubleclick.net *and*google-analytics.com*, were tracking all the 25 devices in our dataset.*

“organizations contacted”, we performed whois lookups on all servers contacted and mapped them to an organization name, allowing us to tighten our upper bound on the number of companies/entities able to track the user through a single app. Returning to the figure, we see... **??**...[**TBD: Amy...**]

iPhone Apps. [**TBD: Shen...**]

6.3 Personally Identifiable Information

Finally, we turn to information leaked by individual applications. We do not report on data leaked for our real users here, but only the data leaked by our controlled apps in isolation. We created fake user accounts on the test phones for a fake user named “Tess Droid”, with fake contact information and fake Twitter and Facebook accounts. We were then able to check that none of this data ever was released over the

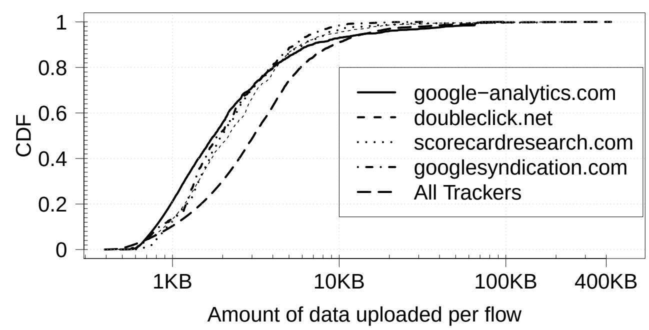

Figure 8: Distribution of bytes uploaded by ads and analytics sites. *The distribution of bytes uploaded by all ads and analytics sites and the top four ads sites based on traffic volume across all users.*

network, either in plaintext (HTTP) or encrypted (HTTPS, see §**??**).

We consider data to be ‘leaked’ when any personally identifiable information – email address, phone number, IMEI number – is sent across the network under HTTP or HTTPS. Some of this information may be relevant to the app – *e.g.*, many apps legitimately require email access. However, none of this information should ever travel across the network in plaintext (HTTP), which we see violated in serveral cases.

In Table 4, we see the type of PII leaked for both Android and iPhone apps. For Android apps, IMEI and Android ID are the most commonly leaked forms of PII in both the Google Play and third-party dataset. Although not popularly thought of as “private” data, each of these identifiers are globally unique: IMEI is a unique identifier tied to a phone, and an Android ID is an identifier tied to a user’s Google Account, used across many services on the Internet. Consequently, either of these datapoints can be used to track or correlate a user’s behavior across all sites the user visits that sell or collaborate with tracking data: a user’s behavior on one site can easily be linked to their behavior on any other site they visit. With Android ID being tracked by between 10 and 20% of apps in our study, and IMEI being tracked by between 5% and 13% of apps in our study, this suggests that global user tracking across collaborating services can be easily achieved today just by using this identifier. [**TBD: ...**]

Other informaiton like contacts, email, and passwords were rarely leaked in the clear, but all were leaked on occaision, suggesting that stricter monitoring of Android app behav-

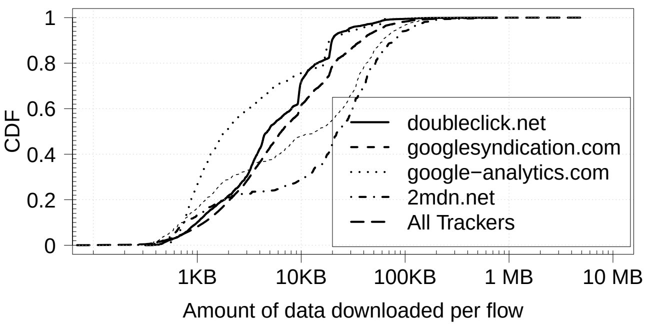

Figure 9: Distribution of bytes downloaded by ads and analytics sites. *The distribution of bytes uploaded by all ads and analytics sites and the top four ads sites based on traffic volume across all users.*

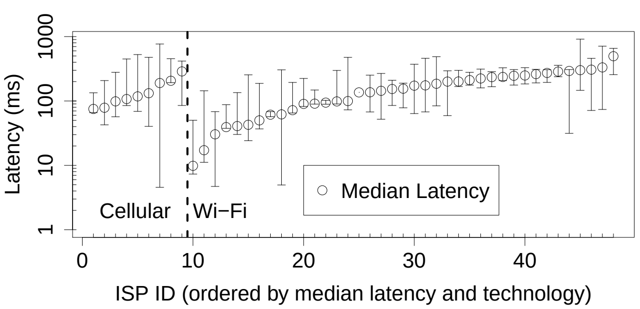

Figure 10: One-way latency from VPN server to mobile devices. *Connections from cellular ISPs suffer a higher delay compared to Wi-Fi ISPs. The delays from Cellular ISPs is comparable to connecting from a Wi-Fi ISP in another country. Error bars indicate the 91st and 9th percentile.*

ior is needed – contrastingly, no iPhone apps (which are manually given clearance by Apple before hitting the iPhone store) leaked passwords in plaintext [**TBD: is this true.**]

Moving to iPhone apps, [**TBD: ...**]

7. BEHAVIOR OF NETWORKS

7.0.1 Controlled Experiments

7.0.2 In the Wild

We ignore connections from the same network and ISP in which our servers were placed.

[**TBD: We performed a traceroute from our server to the egress link and found**]

8. RELATED WORK

Placeholder

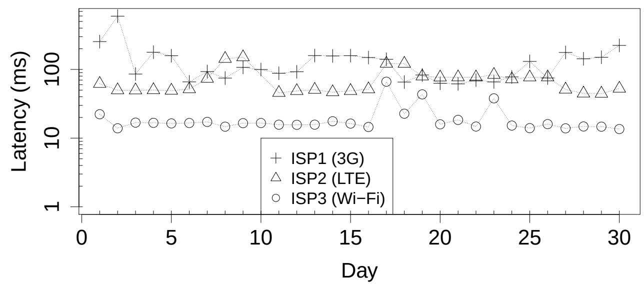

Figure 11: Comparison of ISPs that serve the same user during a 30 day time period. *The LTE service provider has a smaller latency to the 3G provider. The smallest latency is observed by in the home Wi-Fi network.*

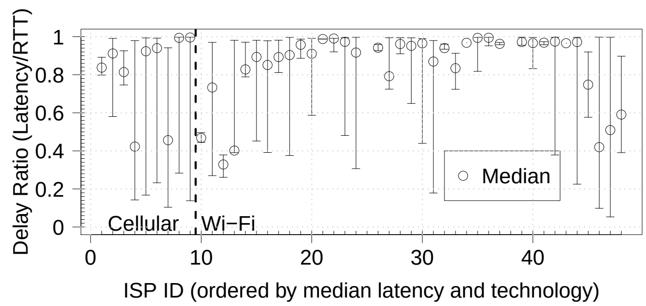

Figure 12: Latency as a fraction of the round trip time to contact google services. *In 35 ISPs of the 48 ISPs we observe that the latency of the mobile device to our server accounts for more than 90% of the end-to-end round trip time. Error bars indicate the 91st and 9th percentile.*

9. CONCLUSION

Placeholder

10. REFERENCES

[1] M. Egele and C. Kruegel. PiOS: Detecting privacy leaks in iOS applications. *Proc. of the NDSS Symposium*, 2011.

[2] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proc. of USENIX OSDI*, 2010.

[3] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of CCS*, 2011.

[4] G. Maier, F. Schneider, and A. Feldmann. A First Look at Mobile Hand-held Device Traffic. *Proc. PAM*, 2010.

[5] L. Ravindranath and J. Padhye. AppInsight: Mobile

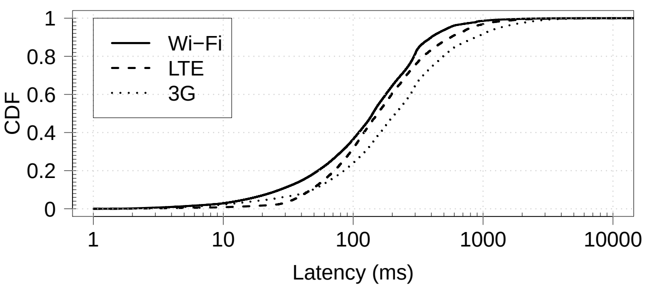

Figure 13: Distribution of latency over cellular and Wi-Fi ISPs. *The distribution of latency observed when using LTE in the wild is similar to that observed for Wi-Fi .*

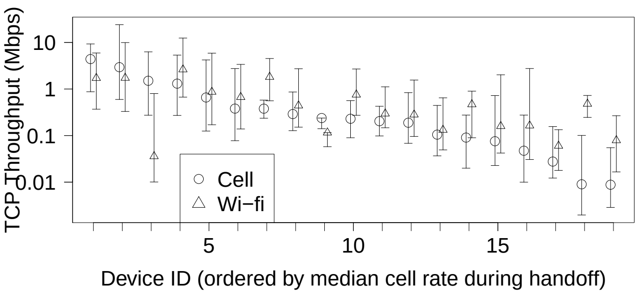

Figure 14: TCP Throughput observed during the hour of the handoff. *The three users that have LTE connections observed a better TCP throughput over LTE in comparison to Wi-Fi in the hour of the handoff. Error bars indicate the 91st and 9th percentile*.

App Performance Monitoring in the Wild. *Proc. of USENIX OSDI*, 2012.

[6] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *Proc. of USENIX NSDI*, 2008.

[7] Strongswan. `www.strongswan.org`.

[8] N. Vallina-Rodriguez, J. Shah, A. Finamore, H. Haddadi, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proc. of IMC*, 2012.