# 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 – compromising network coverage. In this work, we are the first to present an approach that compromises none of these, potentially enabling a large-scale deployment and comprehensive view of mobile Internet traffic across carriers, devices and access technologies.

This paper is the first to explore the opportunities for mobile traffic 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 *MobiScope* 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 open source and publicly available by the time of publication.

- 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

The network behavior of mobile systems has implications for battery life, data-plan consumption, privacy, security and performance, among others. When attempting to characterize this behavior, researchers face a number of trade-offs: compromising network coverage (limiting the number and type of ISPs measured), portability (limiting the device OSes) and/or deployability (limiting subscriber coverage). *MobiScope* compromises none of these, enabling comprehensive measurements across carriers, devices and access technologies. Table **??** puts our approach in context with previous approaches for measuring the network behavior of mobile systems.

Traces from mobile devices can inform a number of interesting analyses. Previous work uses custom OSes to investigate how devices waste energy [8], network bandwidth and leak private information [3, 5]. Similarly, AppInsight [10] and PiOS [2] can inform app performance through binary instrumentation and/or static analysis. In this work, we explore the opportunity to use network traces alone to reveal these cases without requiring any OS or app modifications.

Network traces from inside carrier networks provide a detailed view for large numbers of subscribes. For example, Vallina-Rodriguez *et al.* [13] uses this approach to characterize performance and the impact of advertising. Gerber *et al.* [4] similarly use this approach to estimate network performance for mobile devices. [7] [1] Similar to these approaches, *MobiScope* provides continuous passive monitoring of mobile network traffic; however, *MobiScope* is the first to do so across all networks to which a device connects.

Last, active measurements allow researchers to understand network topologies and instantaneous performance at the cost of additional, synthetic traffic for probing. In contrast, *MobiScope* uses passive measurements to characterize the traffic that devices naturally generate.

## 3. MOBISCOPE OVERVIEW

In this section, we present the *MobiScope* plateform whose goal is *to monitor all the Internet traffic from and to mobile devices* with the following constraints.

1. *OS agnostic.* We want to monitor traffic independently of the OS run by the monitored device. In particular, we do not want to develop OS specific applications, or to root or jailbreak the phone.
2. *ISP agnostic.* We want to monitor traffic without any support from ISPs and cellular providers.
3. *Access technology agnostic.* We want to monitor traffic whatever the access technology used by the mobile device (Wifi, GSM, CDMA, UMTS, LTE, etc.)
4. *Encryption agnostic.* We want to monitor traffic both in clear and encrypted.
5. *Flow modification.* We want to not only monitor, but also possibly modify data packets in order to experiment. This makes *MobiScope* both a passive monitoring and an

| | Network Coverage | Portability | Deployment model | Meas. Type |
|---|---|---|---|---|
| AT&T/Telefonica study [13, 4] | Single carrier | All OSes | Instrument cell infrastructure | Passive |
| WiFi study [1] | Single WiFi network | All OSes | Instrument WiFi network | Passive |
| PhoneLab/TaintDroid [3] | Multiple networks | Android | Install custom OS | Active/Passive |
| MobiPerf [14]/SpeedTest [11] | 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.

experimental platform.

6. *Single machine.* We want *MobiScope* to fit on a single machine to facilitate its installation and deployment.

At the time of the camera ready version, we will make the *MobiScope* software publicly available.

## 3.1 MobiScope Design

In order to monitor all Internet traffic from and to mobile devices with the constraints we described, we designed the *MobiScope* plaform on a VPN infrastructure. By instrumenting the VPN server that mobile devices are using, we can monitor all the Internet traffic going through the VPN tunnels. But, using a VPN infrastructure raises three important questions. i) How ubiquitous is the VPN technology on mobile devices? ii) How to monitor traffic on *MobiScope*? iii) How to modify traffic on *MobiScope*? We explore in the following these three key questions.

### *3.1.1 VPN Technology on Mobile Devices*

The VPN technology is widely supported on the most popular mobile OSes. Indeed, Android, BlackBerry, Bada, and iOS all support VPNs, primarily to satisfy their enterprise clients that use VPNs to securely connect to their enterprise networks. Also, the VPN technology on mobile devices encapsulates all the data traffic—both Wi-Fi and cellular. Consequently, it is possible to reach our three first design goals: OS, ISP, and access technology agnostic.

However, to capture all Internet traffic from and to a mobile device, a VPN must always be enabled. Currently, all iOS devices (version 3.0 and above) support a feature called *VPN On-Demand*. VPN On-Demand forces the iOS device to use VPN tunnels when connecting to a specified set of domains, and we configured it to be enabled for all Internet domains. Android version 4.2 and above support an *Always On VPN* connection that is always enabled for all data traffic, and Android version 4.0 and above provide an API that allows applications to manage VPN tunnels. We implemented an application that uses this API in order to provide the always on functionality for Android 4.0 and 4.1.

In addition, to facilitate the creation of certificates to configure the VPN tunnels between the mobile devices and the *MobiScope* platform, we created our own *MobiScope* root certificate that is used to sign all subsequent certificates issues to mobile devices that want to use the *MobiScope* platform.

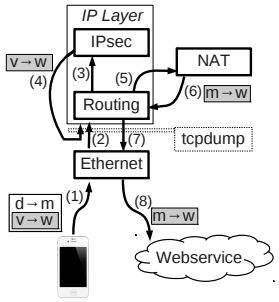
In summary, the VPN technology is available out-of-the-box for the latest versions of the most popular mobile OSes. From the server side, that is on the *MobiScope* platform, we use *Strongswan* [12], an open source software used to manage VPN tunnels. Strongswan supports the required protocols and encryption algorithms required by iOS and Android, in particullar IPsec for the encryption of IP datagrams and IKEv1 and IKEv2 [6] for authentication and key negotiation.

### *3.1.2 Monitoring Traffic on MobiScope*

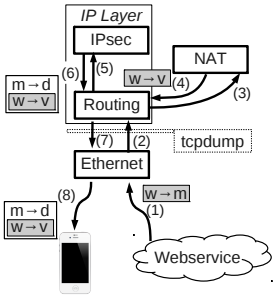
Surprisingly, monitoring traffic on *MobiScope* is particularly involved. When we monitor traffic on *MobiScope*, we need to dump all traffic between the mobile devices and the Internet, and we need to associate the dumped traffic to the right mobile device and Webservice. The former is easy using a *tcpdump* process, but the later is more complex. To explain this complexity, we first need to explain how packets are processed on *MobiScope*.

Figure 1 (a) shows the processing on *MobiScope* of packets sent from a mobile device to the Internet. At step (1), (2), and (3) the encrypted datagram (in gray) goes up the layers, this datagram is encapsulated in a regular IP datagram sent from the device (address $d$) to the *MobiScope* box (address $m$). The IPsec layer decrypts at step (3) the datagram whose source IP address is the private address of the mobile device in the VPN tunnel (address $v$) and the destination is the IP address of the destination Webservice (address $w$). In order to send this datagram in the Internet, the NAT must convert in step (5) the private IP address $v$ to the public IP address $m$ of the *MobiScope* box. Then this packet is sent in the Internet. As the tcpdump process runs just above the Ethernet layer, it dumps packets at step (2), (4), and (7). In addition, we also dump the NAT translation table. To associate packets to a device and a Webservice, we only need the packet captured at step (4) that associates the packet to the private address $v$ of the device in the VPN tunnel and the Webservice (address $w$), and the NAT translation table that gives the mapping between the device in the VPN tunnel (address $v$) and the public IP address $d$ of the mobile device.

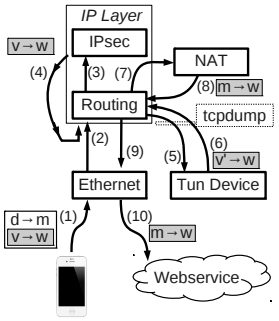
However, in the reverse direction, when packets flows from the Internet to the mobile device, it is no more possible to associate a mobile device to the packets, see Figure 1 (b). Indeed, from the dumped packets at step (2) and (7), we know that a packet in sent by the Webservice to the *MobiScope*



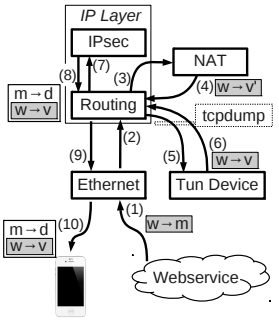

(a) Packet from mobile device. *Tcpdump can capture packets at step (2) $d \rightarrow m$, (4) $v \rightarrow w$, and (7) $m \rightarrow w$.*




(b) Packet to mobile device. *Tcpdump can capture packets at step (2) $w \rightarrow m$ and (7) $m \rightarrow d$, however it is cannot log the packet $w \rightarrow v$.*




(c) Packet from mobile device. *Tcpdump monitoring packets on the tun device can capture packets at step (5) $v \rightarrow w$, and (6) $v' \rightarrow w$.*




(d) Packet to mobile device. *Tcpdump monitoring packets on the tun device can capture packets at step (5) $w \rightarrow v'$, and (6) $w \rightarrow v$.*

| Symbol | Description |
|---|---|
| d | IP address of the mobile device assigned by its ISP. |
| m | IP address of the *MobiScope* server. |
| w | IP address of the server providing the Webservice. |
| (i) | The i-th step of packet processing. |
| a → b | Packet with source IP *a* and destination IP *b*. |
| v | IP address of the mobile device in the VPN tunnel. |
| v' | Temporary IP address of the mobile device used to send the packet to the TUN device. |

Figure 1: Packet monitoring in the *MobiScope* box.

box (step (2)), but then this packet is encapsulated at the IPsec layer, and the address resolution is performed by the NAT without any dump. So, when we see the datagram at step(7), we have no way to know which encrypted packet is encapsulated. We need to dump the packet at step (4), but we have no access to it. One way would be to run the NAT on a separate box, and to dump traffic from that box, but it would require two different boxes. Instead, we propose a solution that allows to dump the packet from *w* to *v* from a single box, which we describe in the following.

Our solution is to force the packets from *w* to *v* to make a loop on a TUN device, and to dump these packets at the TUN device, see Figure 1 (d). Indeed, when a packet is received from the Internet is goes up the layers up to the NAT at step (3). At that point, the NAT resolve the public address of the *MobiScope* box *m* to the address of the device in the VPN tunnel *v*. In the NAT all addresses in the VPN tunnel are class A private addresses using only 23 bits out of 24. The 24th bit has a specific role. By default it is set to 0, but we modified the NAT so that when it receives a packet that resolve to *v*, it changes it to *v'* that only differ from *v* by the 24th bit that is set to 1. The only one reason to make this conversation is to send all packets whose destination is *v* to the TUN device, by creating a specific forwarding rule that forwards all packet with a *v'* destination address (a class A private address with the 24th bit set to 1) to the TUN device, step (5). Then, we implement a process at the TUN device whose only one goal is to change the destination address from *v'* to *v*, and to sends back the packet to the IP layer, step (6). Then the packet follows the path of a regular packet in a VPN tunnel.

When a packet is received from the mobile device, we perform a similar process that is described in Figure 1 (c).

In summary, exploiting the notion of TUN device, we are now able to dump all Internet traffic from and to a mobile device connected to the *MobiScope* platform from a single machine, and to associate this traffic to corresponding mobile devices and Web services.

### *3.1.3 Modifying Traffic on MobiScope*

We described in section 3.1.2, how we can monitor all Internet traffic on the *MobiScope* platform, using the notion of TUN device. In that section, we explain here how we can use this same notion of TUN device to build a powerful plugin infrastructure enabling any traffic modification.

Figure 2 shows th

## 3.2 Feasibility

## 3.3 Limitation

*MobiScope* has reached all the design goals, making this platform an excellent choice to monitor traffic for mobile devices. However, *MobiScope* has one limitation: because all the traffic between mobile devices and *MobiScope* is encrypted, we cannot observe any traffic modification made by

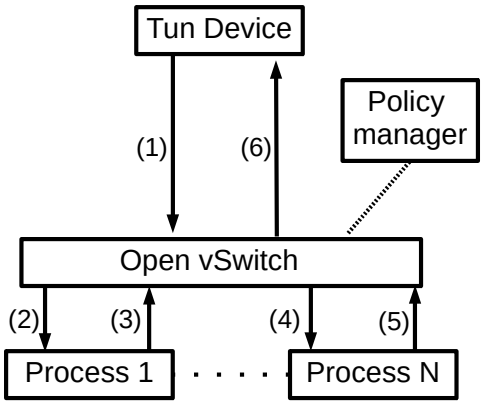



Figure 2: Plugin Infrastructure on *MobiScope*.

ISPs. Indeed, ISPs that modify traffic (e.g., to insert advertisements [**TBD: AR: give a reference**]) must perform deep packet inspection, which is not possible when the traffic is encrypted. In this section, we assess to which extent this issue is a practical limitation for *MobiScope*.

[**TBD: AL: put tripwire experiments here to show that ISP traffic modification does not impact much Mobiscope.** ]

## 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.

### 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 [7, 13], 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?

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

| 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.*

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 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%

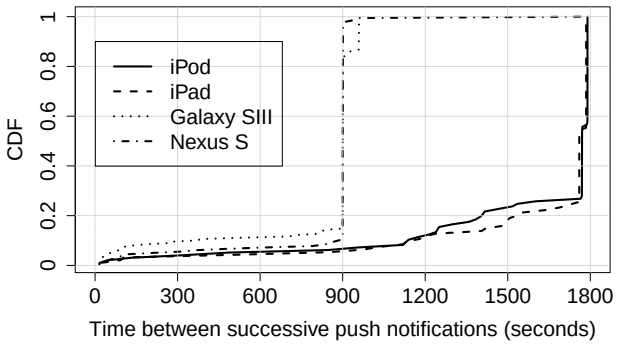



Figure 3: Inter-arrival time between push notification messages after factory reset. *The iOS devices communicate with the notification server approximately once every 1800 seconds while the Android devices communicate once every 900 seconds with their notification server. [**TBD: Add results from shen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

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

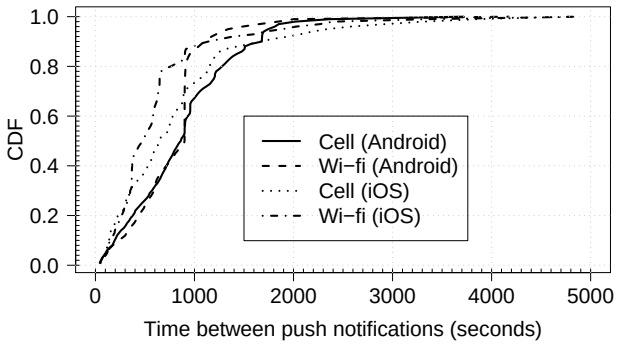



Figure 4: Distribution of the time between push notification messages in the wild. *The frequency of push notification messages is higher for the iOS devices in our dataset compared to the Android devices. Notification messages are less frequent over cellular networks compared to Wi-Fi networks.*

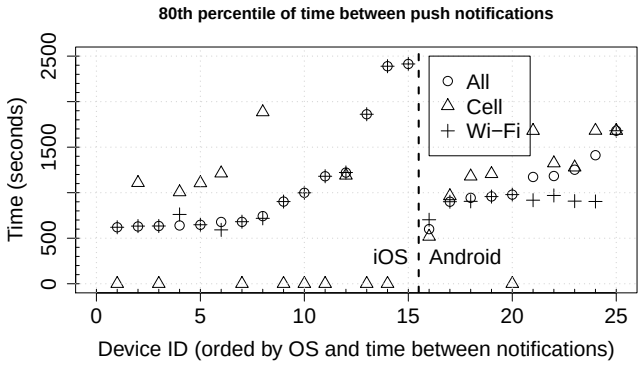



Figure 5: Inter-arrival time between push notifications in the wild. *Push notifications occur less frequently over Cellular networks. The rate of push notifications depends on users and the applications installed. [***TBD: DISCUSS: Better representation for tablets – currently they have a value of 0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

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 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

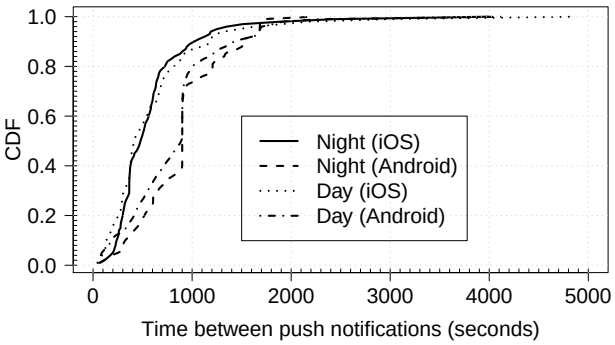



Figure 6: Impact of time-of-day on the push notifications. *The rate of push notifications is agnostic of the time of the day for iOS devices.*

[**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...**] We see that in general, more bytes are received than sent (289711536 vs 3040151). This is probably due to advertising, where the size of the requests is much smaller than the advertisements received. Indeed, most of the applications contacted Google, through AdMob. A few third-party advertising servers were also contacted.

Applications in general fit their expected bandwidth usage(Spotify, predictably sent the most bytes (sent 318980 received 12108538) and a notepad application used the least (sent 414 received 262). However, some games used a surprisingly large amount of bandwidth (The Simpsons(TM): Tapped Out 149291 88932890 and Jurassic Park(TM) Builder 312912 127163147 (These two received the most bytes)). Ludia games in particular used an unusually large amount of bandwidth. Ludia's Jurassic Park(TM) Builder ran for 13 minutes, and Ludia's Family Feud and Friends Free ran for 7 minutes. Ludia's Jurassic Park(TM) Builder was 38.1 MB and Ludia's Family Feud and Friends Free was 27.6 MB. This may be because of game updates (for example, a lot of these games may have released more in-game content for Easter, new textures, new models, etc.) This is confirmed by the fact that most of the data transmitted is near the beginning. These applications were tested before Google updated its Play Developer Program Policies on May 1, 2013 to say, "an app downloaded from Google Play may not modify, replace or update its own APK binary code using any method other than Google Play's update mechanism." However, companies may still subvert this new policy by pushing textures and other in-app purchases aren't necessarily modifying the apk binary code, but still use excessive bandwidth.

In terms of categories, games had the most variable amount of usage. For example, The aforementioned Jurassic Park sent (com.ludia.jurassicpark 312912 127163147), but Angry Birds StarWars only had (com.rovio.angrybirdsstarwars.ads.iap 5463 9416) Fruit Ninja only had (com.halfbrick.fruitninjafree 4326 28639). Music/video applications were consistently the biggest users of bandwidth (com.spotify.mobile.android.ui 318980 12108538 ), (com.pandora.android 87753 7744094), (com.oovoo 148905 1402490). ooVoo is a video chat and I'm application. Pinterest, through sheer number of pictures, also sent and received a lot. (com.pinterest 156963 3309776) Productivity/tool apps tended to use the least (com.socialnmobile.dicta 414 262),(com.sirma.mobile.bible.android 2255 6523) .

Methodology: Each pcap file is associated with an application. Simply look through all packets sent/received and find their lengths. Sum to get bytes sent and received. Script to do this is in totalBytes.sh. To get size of packets over time, simply remove the summing part. This script is in bytesSentOverTime.sh Unfortunately, due to the usage of many computers with differing timestamps while working with the hundred pcaps, these need to be manually transcribed from pcap to app name. file_to_appname.txt has a rough approximation. Use map_file_to_name.rb file_to_appname.txt <file>. For the third party apps though, these were all conducted on a single computer, so just simply use the map_results_to_app. script. (If conducting these experiments in the future, would recommend synchronizing the time of the computer(s) running the experiment and the time of the thing handling the pcap-captures)

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 "organizations contacted", we performed whois lookups on all servers contacted and mapped them to an organization

| 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.

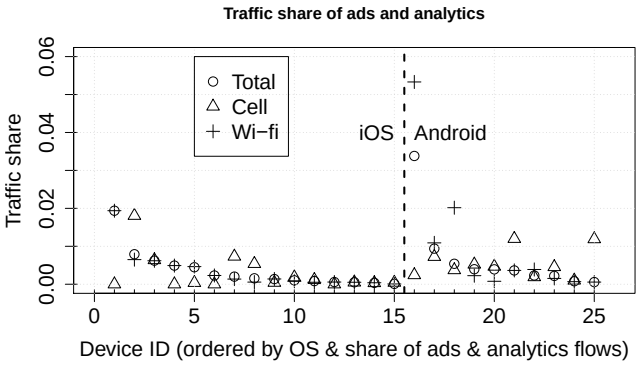



Figure 7: Fraction of traffic volume because of Ads and Analytics. *[**TBD: Check for id1 and id25**]*

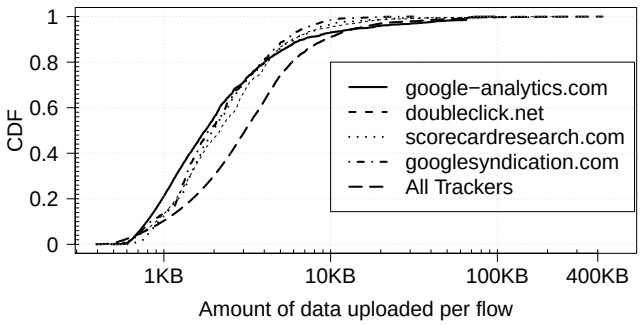



Figure 8: Distribution of bytes uploaded by ads and analytics sites. *The distribution of bytes uploaded by all ads and analytics sites and the top four ads sites based on traffic volume across all users.*

| **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.*

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...**] After doing whois lookups on all contacted servers, we categorized by hand each server. For all of the hundred apps, we broke down the 51 servers that each visited into these categories:

- 3 Advertising
- 1 Analytics
- 6 CDN
- 10 Hosting
- 1 IANA
- 1 ISP
- 3 Network
- 23 PrivateCompany
- 3 RegionalInternetRegistry

For all of the third party apps, we broke down the 96 servers that each visited into these categories:

- 4 Advertising
- 1 Analytics
- 8 CDN
- 31 Hosting
- 1 IANA
- 2 ISP
- 8 Network
- 37 PrivateCompany
- 4 RegionalInternetRegistry

Words with Friends contacted an astounding 21 servers:

- 1 Advertising
- 3 CDN
- 7 Hosting
- 1 IANA

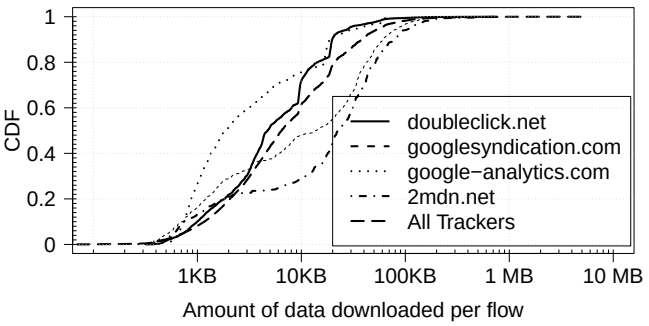



Figure 9: Distribution of bytes downloaded by ads and analytics sites. *The distribution of bytes uploaded by all ads and analytics sites and the top four ads sites based on traffic volume across all users.*

- 1 Network
- 5 PrivateCompany
- 1 RegionalInternetRegistry

ooVoo Video Call also contacted 21 servers, and interestingly, contacted Fastly, an analytics company.

- 2 Advertising
- 1 Analytics
- 5 CDN
- 1 Hosting
- 1 IANA
- 1 ISP
- 1 Network
- 8 PrivateCompany
- 1 RegionalInternetRegistry

Small practical apps such as PhotoGrid - Collage Maker and ColorNote Notepad Notes generally did not contact many servers at all.

- com.socialnmobile.dictapps.notepad.color.note
- 10.11.4.21 Internet Assigned Numbers Authority
- 128.208.4.1 University of Washington
- 74.217.75.7 Internap ”
- com.roidapp.photogrid
- 10.11.4.22 Internet Assigned Numbers Authority
- 128.208.4.1 University of Washington
- 173.194.33.45 Google Inc.”

Comments: wasn't entirely sure how to differentiate between 1. server requested and getting CDNs back and 2. server requested, getting actual server back.

**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 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 behavior 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.

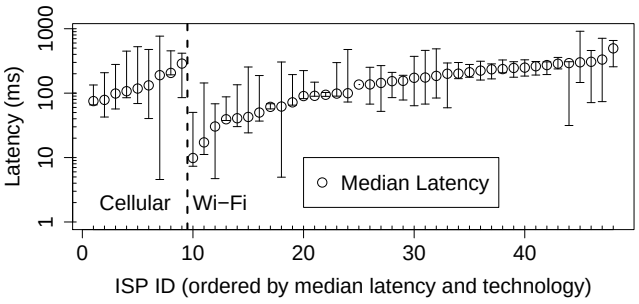



Figure 10: One-way latency from VPN server to mobile devices. *Connections from cellular ISPs suffer a higher delay compared to Wi-Fi ISPs. The delays from Cellular ISPs is comparable to connecting from a Wi-Fi ISP in another country. Error bars indicate the 91st and 9th percentile.*

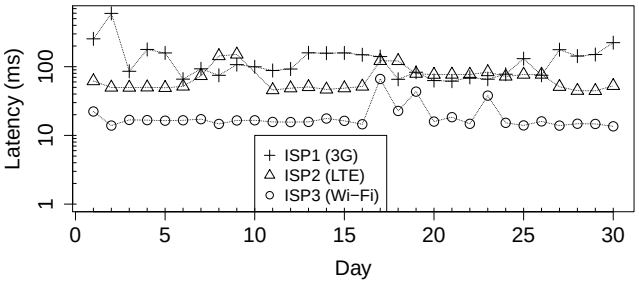



Figure 11: Comparison of ISPs that serve the same user during a 30 day time period. *The LTE service provider has a smaller latency to the 3G provider. The smallest latency is observed by in the home Wi-Fi network.*

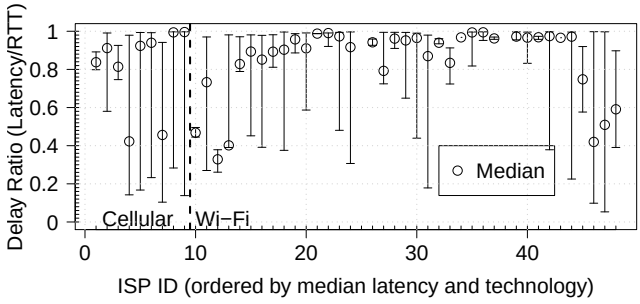



Figure 12: Latency as a fraction of the round trip time to contact google services. *In 35 ISPs of the 48 ISPs we observe that the latency of the mobile device to our server accounts for more than 90% of the end-to-end round trip time. Error bars indicate the 91st and 9th percentile.*

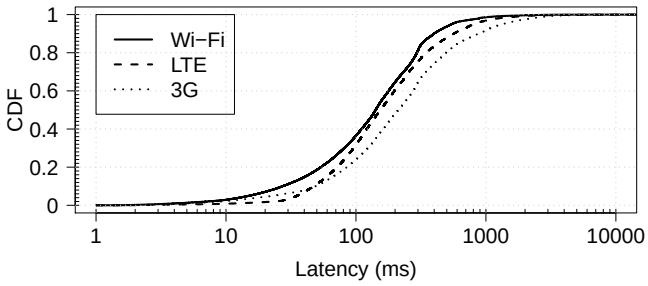



Figure 13: Distribution of latency over cellular and Wi-Fi ISPs. *The distribution of latency observed when using LTE in the wild is similar to that observed for Wi-Fi.*

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

## 8. RELATED WORK

Placeholder

## 9. CONCLUSION

Placeholder

## 10. TRAFFIC CLASSIFICATION

This is a raw dump of the results. I have to work on putting this in the context of iOS vs Android.

1. Traffic classification: Is it possible to identify the source of the traffic by just looking at the traffic? What heuristics can be used? What fraction of the traffic can be classified to specific webservices?

2. Traffic characteristics: What are the network traffic characteristics of popular webservices/apps? Impact of the access technology and operating system on the webservices?

3. Information exchange: How are CDNs and other servers used? What fraction of traffic is due to ads and analytics and other trackers? How frequently is PII leaked and to which hosts?

Use well known port numbers to broadly categorize tcp and udp traffic. TCP traffic is classified as either HTTP, SSL, or other while UDP is classified as either DNS or other. SSL: 443/tcp, 563/tcp, 585/tcp, 614/tcp, 636/tcp, 989/tcp, 990/tcp, 992/tcp, 993/tcp, 995/tcp, 5223/tcp. HTTP: 80/tcp, 81/tcp, 631/tcp, 1080/tcp, 3138/tcp, 8000/tcp, 8080/tcp, 8888/tcp. Traffic such as ICMP which is neither TCP or UDP is classified as other. In Table 5 we observe that based on this simple classification we observe that more than 90% of the traffic is either HTTP or SSL. We observe that the traffic share of SSL over cellular is more than twice the traffic share observed over Wi-Fi

### 10.1 Classification of HTTP Traffic

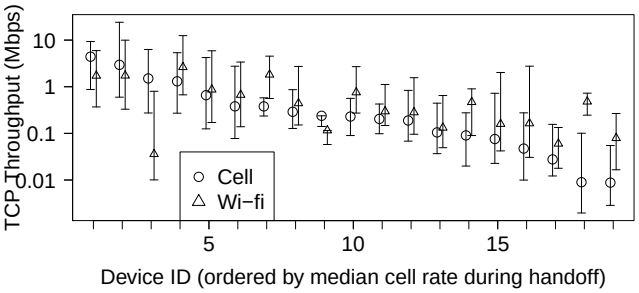



Figure 14: TCP Throughput observed during the hour of the handoff. *The three users that have LTE connections observed a better TCP throughput over LTE in comparison to Wi-Fi in the hour of the handoff. Error bars indicate the 91st and 9th percentile.*

| Protocol | Service | Android | | iOS | |
|---|---|---|---|---|---|
| | | Cell. | Wi-Fi | Cell. | Wi-Fi |
| TCP | HTTP | 35.386 | 68.686 | 52.109 | 75.506 |
| | SSL | 61.134 | 24.366 | 46.765 | 15.777 |
| | other | 2.346 | 6.290 | 0.256 | 1.818 |
| UDP | DNS | 0.681 | 0.496 | 0.545 | 0.305 |
| | other | 0.315 | 0.096 | 0.283 | 6.580 |
| Other | - | 0.134 | 0.062 | 0.039 | 0.011 |
| *total* | | 100.00 | 100.00 | 100.00 | 100.00 |

Table 5: Traffic volume (in percentage) of popular protocols on Android and iOS devices over cellular and Wi-Fi. [**TBD: Verify total 100 for final results**] *Traffic share of SSL over cellular networks is more than twice the traffic share of SSL over Wi-Fi.*

A mobile application can append information about the application, carrier, and other information related to the device in the User-Agent field. Webservices use this information to segregate traffic from the mobile devices and carriers and to offer custom services. Recent studies on mobile traffic classification have used the User-Agent field to classify mobile HTTP traffic [9, 7, 15]. We now provide a comparison on the usefulness of the User-Agent field when classifying mobile traffic and discuss the issues with Android devices and media traffic.

We observe that more than 98% of HTTP traffic from the iOS and Android devices have a valid User-Agent string. [**TBD: Figure 15**]). In the 95.96 GB of data traffic, we observed a total of 1435 unique User-Agent strings. We extract the application identifiers in the User-Agent strings by using regular expressions that replace the OS, carrier, and other auxillary information such as delimiters with a blank character. We then cluster these blank separated tokens based on the number of similar tokens. We further cluster these

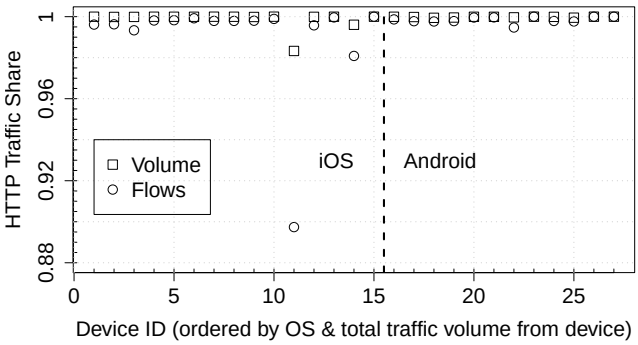



Figure 15: HTTP traffic with a User-Agent.

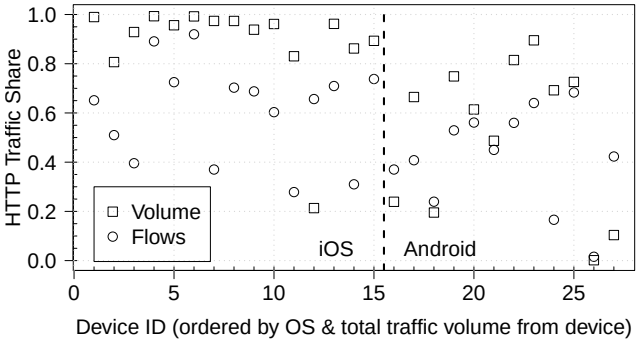



Figure 16: HTTP traffic with a User-Agent with an identifier of an application or OS service.

clustered tokens using the edit distance. At the end of this process we were able to identify 361 unique application signatures. We use the extracted application signature to label the HTTP traffic that flows through *MobiScope*.

In Figure 16 we plot the fraction of HTTP traffic for which an application signature for found; the devices are ordered according to the operating system, and for each operating system we further order the devices according to the total traffic from the device that flowed through *MobiScope*. We observe that a significantly larger fraction of traffic from iOS device can be mapped to an application in comparison to the traffic from Android devices. For example, while more than 80% of HTTP traffic from iOS device was labeled to an application, we were able to classify only 23.9% of 9.6 GB of HTTP traffic from device 16 and 19.5% of the 2.6 GB of HTTP traffic from device 18. This difference is because of the techniques used by Android and iOS application to download audio and video content.

The iOS devices use the AppleCoreMedia service to download audio and video content. For example, we observed that AppleCoreMedia was mentioned in the User-Agent string for 98.45% of the content downloaded from the YouTube servers. We classify YouTube servers based on the host name specified the HTTP GET requests. We observed that AppleCoreMedia was the primary application that fetched content from other media sites such as Netflix and iTunes. Like

| User-Agent | OS |
|---|---|
| AppleCoreMedia/1.0* | iOS |
| stagefright/1.2* | Android |
| Dalvik/1.6* | Android |
| Linux; Android | Android |
| com.google.android.youtube* | Android |

Table 6: User agents while streaming youtube videos. *We use * for the characters not shown due to lack of space.*

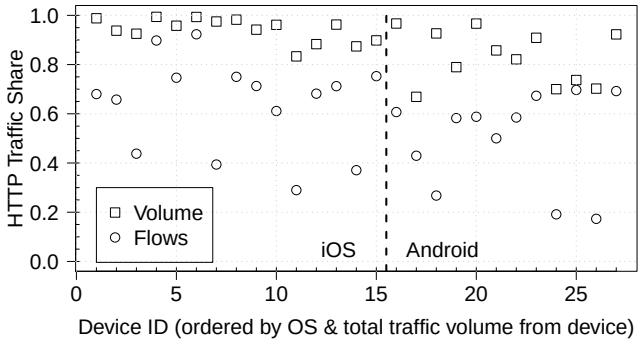



Figure 17: HTTP traffic classification using User-Agent or name of media service.

iOS devices have AppleCoreMedia, Android provides Stagefright[**?**] as media library. However, only 41.91% of YouTube content in our dataset was fetched using Stagefright in the User-Agent field. In contrast, 56.52% of YouTube content was downloaded by Android devices without any application signature in the User-Agent field.

To further classify the media content, we use the *HOST* field in the HTTP GET requests. We assign flows a label based on the host names of popular media sites. In Figure 17 we plot the fraction of traffic that we could classify by using the User-Agent along with the host names. We observe that despite this technique the traffic share of classified traffic for Android devices still lags behind that of iOS devices.

What we show Difference in media content download in iOS and Android The fields in the HTTP requests that can be used to classify HTTP content and identify traffic sources. The difference in the use of UserAgents in Android and iOS.

## 10.2 Classification of SSL Traffic

Field observed in traffic: server name and the Use of subject field in the certificates. Issues with generic certificates CN=*.google.com. Workaround: Use of DNS

[1]

## 10.3 Misc

# 11. REFERENCES


[1] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network performance of smart mobile handhelds in a university campus wifi network. In *Proc. of IMC*, 2012.

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

[3] 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.

[4] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman. Speed testing without speed tests: estimating achievable download speed from passive measurements. In *Proc. of IMC*, 2010.

[5] 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.

[6] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Rfc 5996: Internet key exchange protocol version 2 (ikev2). *IETF Request For Comments,* `http://www.ietf.org/rfc/rfc5996.txt`, 2010.

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

[8] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof. In *Proc. of Eurosys*, 2012.

[9] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones. In *Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12*, pages 127–140, New York, New York, USA, June 2012. ACM Press.

[10] L. Ravindranath and J. Padhye. AppInsight: Mobile App Performance Monitoring in the Wild. *Proc. of USENIX OSDI*, 2012.

[11] J. Sommers and P. Barford. Cell vs. wifi: On the performance of metro area mobile connections. In *Proc. of IMC*, 2012.

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

[13] N. Vallina-Rodriguez, J. Shah, A. Finamore,


| **Hostname** | **Number of devices** | |
|---|---|---|
| | **iOS** | **Android** |
| www.facebook.com | 11 | 8 |
| b.scorecardresearch.com | 10 | 5 |
| p.twitter.com | 7 | 1 |
| www.google-analytics.com | 6 | 2 |
| bcp.crwdcntrl.net | 4 | 3 |

Table 7: The top 5 hosts accessed from Android and iOS devices with an empty user agent field

H. Haddadi, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proc. of IMC*, 2012.

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

[15] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proc. of IMC*, page 329, New York, New York, USA, Nov. 2011. ACM Press.