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
Northeastern University

ABSTRACT

Characterizing Internet traffic naturally generated by mobile devices is an open problem because mobile devices and their OSes provide no built-in support to monitor network traffic. Therefore, researchers exploring the mobile traffic in the Internet either work on real network traces, but without the possibility to control the experiment, or on custom OSes requiring to root or jailbreak the devices, but with the difficulty to scale the experiment to a large number of various devices.

In this paper, we take an alternative approach: monitoring through indirection. Specifically, we exploit the fact that most mobile OSes support proxying via virtual private networks (VPNs). Sending mobile Internet traffic through a proxy server under our control enables us to monitor all flows regardless of device, OS, or access technology. We argue that our solution, *MobiScope*, has reasonable overheads and can be configured on existing phones without any OS modification. This makes *MobiScope* feasible for a large variety of experiments from a small scale controlled experiment to a large-scale experiment with a large variety of devices, OSes, and cellular providers.

We present the architecture of *MobiScope*, a software package running on a single machine that can monitor all mobile traffic, and that provides a convenient plugin infrastructure to analyze and modify the mobile traffic on-the-fly. In particular, we present a SSL bumping module that can decrypt and uncover most of the SSL traffic. Then, using *MobiScope* on both controlled experiments and a 7-month IRB-approved in-the-wild study with a small set of real users, we analyze key characteristics of iOS and we compare them with Android, such as the push notification services or the applications network footprint.

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? variety of options available, different access technologies, data plans, news OSes, new versions of applications, decreasing quota**]

The key challenge is that mobile devices and their OSes provide no built-in service for monitoring and reporting *all network traffic*. We strongly believe that a comprehensive network usage analysis must not be limited to specific mobile OSes, access technology, device manufacturer, installed applications, and user behavior. Previous works miss out on at least one of the above dimensions of mobile Internet traffic [21, 11, 5, 8, 22, 19], thus provides 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, applications versions, 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 7-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 either compressed, or encrypted, thus 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**].
- [**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.**]

2. MOBISCOPE OVERVIEW

In this section, we present an overview of the *MobiScope* platform[1]. The goal of *MobiScope* is straightforward: we seek to enable passive monitoring of *all the Internet traffic from and to mobile devices*. While previous work has accomplished this for a limited set of devices or networks, we seek to avoid such limitations.

1. *OS agnostic.* Monitor traffic independently of the OS run by the monitored device. In particular, we avoid the need to develop OS-specific applications, or to root or jailbreak the device.
2. *ISP agnostic.* Monitor traffic without any support from ISPs and cellular providers.
3. *Access technology agnostic.* Monitor traffic whatever the access technology used by the mobile device (Wifi, GSM, CDMA, UMTS, LTE, etc.)
4. *Continuous.* Monitor traffic continuously, even when devices switch between networks or return from being idle.

   [**TBD: There is not more reference to the flow modification (admittedly a bad term) feature. Don't we want to put it as a design goal. Section "Interposing on Traffic on Mobiscope" is just about that. May be we can add the goal "Traffic interposition".**]
5. *Scalability. MobiScope* should be equally feasible to deploy on a single machine or using a collection of VMs in a hosted/cloud deployment.
6. *Encryption agnostic.* Achieve visibility of both encrypted and plaintext traffic.

In the following, we describe in detail the design of *MobiScope*, then we discuss the limitations of the platform.

2.1 MobiScope Design

To meet the above goals, we exploit the observation that nearly all devices support network traffic indirection via virtual private networks (VPNs). In particular, instead of using the VPN server to access a private network, we use it as a proxy for all of a device's mobile Internet traffic. This enables passive monitoring of Internet traffic regardless of device OS, carrier/ISP or access technology. Further, we show that this approach has minimal impact on performance and measurement fidelity – making *MobiScope* a practical approach for pervasive passive network monitoring.

In the following sections, we describe how our measurement infrastructure achieves the goals stated in the previous section. We begin by describing how we addressed several challenges with implementing our VPN-based proxy.

2.1.1 VPNs for Mobile Devices

In this section, we provide a detailed description of how VPNs allow us to achieve many of our *MobiScope* design goals. Our first goal is to provide a measurement system that works regardless of device OS. To achieve this goal, we note that VPNs are widely supported on the most popular mobile OSes. Indeed, Android, BlackBerry, Bada, and iOS all support VPNs, primarily to satisfy their enterprise clients. In this work, we focus on the two most popular OSes: iOS and Android.

Our second and third goals are to enable measurements regardless of ISP or access technology. Fortunately, iOS and Android support VPN connectivity regardless of ISP or access technology – so long as the network supports the Internet Protocol.

We note that both iOS and Android support VPN connections using the IPSec standard, meaning we can implement our VPN proxy server using robust, open-source code from Strongswan [20]. *MobiScope* thus supports IPSec tunnels using either IKEv1 or IKEv2 [**?**] for authentication and key negotiation.

To meet our goal of continuous network monitoring, 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

[1]Our platform is currently in private beta with deployments in the US, France and China. We will make the *MobiScope* software publicly available.

VPN tunnels when connecting to a specified set of domains. To ensure all possible addresses match this list, we observed that iOS uses suffix matching to determine which connections should be tunneled; accordingly, we specified the domain list as the set of alphanumeric characters (a-z, 0-9, one character per domain). 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 support both options: we have distributed Always On VPN configurations and implemented an application that uses the VPN API to provide equivalent functionality for Android 4.0 and 4.1.

2.1.2 Monitoring Traffic on MobiScope

Having shown that VPNs support many of *MobiScope*'s goals, we now describe how we implement passive network monitoring using VPN proxying. While the high-level design for capturing network traffic from mobile devices is straightforward, the implementation is not. In particular, the interactions between IPSec, routing and NAT complicate our ability to map bidirectional flows to individual devices. The following paragraphs describe these challenges and how we addressed them to provide a stand-alone (*i.e.*, single server) mobile-traffic monitoring proxy. We conclude by describing how our architecture facilitates pluggable modules for performing custom monitoring.

At first glance, capturing all traffic traversing a VPN server should be as simple as running a tap on the network interface, *e.g.*using *tcpdump*. While this indeed captures all traffic, it does not capture sufficient information to distinguish bidirectional flows and map them to individual devices. We now describe how to provide this mapping.

First, we describe how the Linux network stack interacts with IPSec. We assume the *MobiScope* server is assigned IP address m and the mobile device's public IP address is d. When the VPN connection is established, the proxy assigns the device a private address v. Last, the device is attempting to access a service located at address w. We denote a packet from source s to destination d as $s \rightarrow d$.

Forward path. The first step is to map flows in the forward direction. Figure 1 (a) shows the path that packets take through a *MobiScope* proxy. At steps (1), (2), and (3) the encrypted datagram (in gray, $d \rightarrow m$) is passed to the IPSec module to decrypt and process the encapsulated IP datagram ($v \rightarrow w$) sent from the device. Because the *MobiScope* assigns private address space to clients, it must use NAT in step (5) to convert the private IP address *v* to the public IP address *m*. Last, the packet is forwarded to the Internet.

We now describe how running tcpdump and tracking the VPN assignment table[2] are sufficient for mapping flows in the forward direction. Running *tcpdump* on the Ethernet device captures packets at step (2), (4), and (7). The assignment table provides a map between the public IP address *d* of the device and its private IP address *v* in the tunnel. To associate packets to a device and a Web service, 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 Web service (address *w*), and the assignment table that gives the mapping between the device in the VPN tunnel (address *v*) and the public IP address *d* of the mobile device.

Reverse path. In the reverse direction (when packets flow from the Internet to the mobile device), it is no longer possible to associate a mobile device to the packets. We refer to Figure 1 (b), where we continue to dump packets from the Ethernet device. From steps (2) and (7), we know that a packet in sent by the service at address w to the *MobiScope* box (step (2)), but then this packet is encapsulated at the IPsec layer – address resolution is performed by the NAT without passing through tcpdump. 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 via the standard Linux networking stack.

Bidirectional mappings. A straightforward solution to the reverse path mapping problem is to forward traffic to a separate NAT device and dump traffic there. To avoid the need for additional hardware/VMs, we virtualize an additional network interface and route traffic through it.

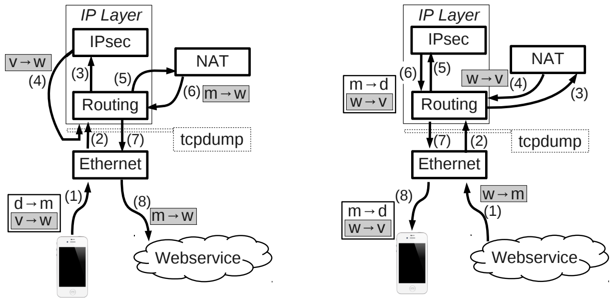
Namely, we use a Linux TUN device, send packets from *w* to *v* through it and run packet captures on that TUN device (Figure 1 (d)). Indeed, when a packet is received from the Internet and arrives at NAT at step (3), the NAT resolves the public address of the *MobiScope* box *m* to the address of the device in the VPN tunnel *v*. We design the NAT such that each client is assigned an address from a space of prefix length $p - 1$. The pth 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 pth bit that is set to 1. This facilitates routing traffic to the TUN device for all packets whose destination is *v*, and facilitates forwarding all packet with a *v'* destination address (pth bit set to 1) to the TUN device, step (5). Then, we implement a process at the TUN device that changes the destination address from *v'* to *v* and sends 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).

2.1.3 Interposing on Traffic on MobiScope

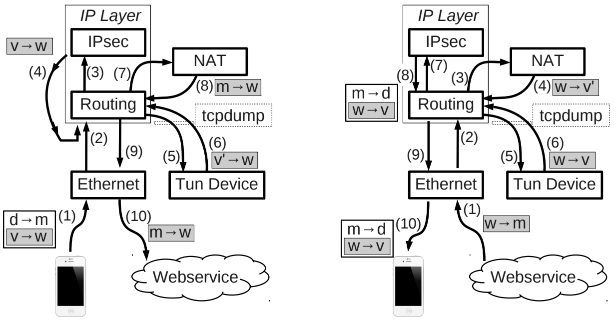
A key limitation of previous approaches to measuring mobile network traffic is that the contents of encrypted (SSL) traffic are unavailable for analysis. Of course, SSL rightfully provides authentication and protects user privacy from eavesdroppers; however, as increasing amounts of Web traffic flow over HTTPS, we lose the ability to understand how

[2]*MobiScope* maps a device identified by its public IP address to a private IP address in the VPN. We call the mapping between this device public and private IP addresses the *VPN assignment table*.



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

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



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

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

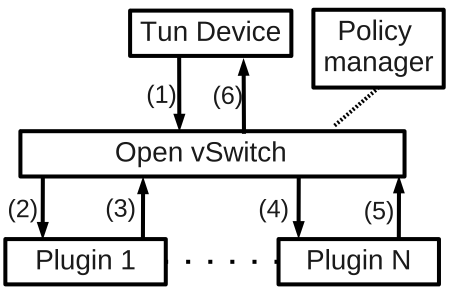


Figure 2: Plugin Infrastructure on *MobiScope*.

to optimize such traffic. This has implications both for performance (page speed optimizations) and privacy (PII leakage over secure channels). In this section, we describe how *MobiScope* allows us to analyze the contents of SSL flows generated by mobile devices.

Plugin infrastructure. Figure 2 shows how we use our virtual network interface (TUN) to support a plugin infrastructure for *MobiScope*. Each plugin takes as input a network flow and outputs a network flow (potentially empty). When a packet is received at the TUN device, it is sent to a software-defined switch [3] that determines the ordered set of plugins that flows will traverse. This order is configured by a policy manager, which determines the set of plugins that should operate on each flow. After the last plugin is traversed, the network flow is sent out through the TUN device to the Internet.

Plugins can be used for many different purposes such as ad blocking, analyzing PII leakage or page speed optimization. In the following, we describe how we use a plugin to enable SSL traffic decryption using the *MobiScope* plugin-infrastructure.

Example plugin: SSL bumping. First, we note that our VPN proxy implementation uses a self-generated *MobiScope* root certificate that is used to sign all subsequent certificates issued to participating mobile devices. This allows us to perform SSL traffic decryption, using the Squid proxy's SSL bumping[**TBD: AR: give a reference**] feature, which is essentially a man-in-the-middle operation on the secure connection.[3] Specifically, when the mobile device connects to a service supporting SSL, the proxy impersonates the service using a forged certificate signed with the root certificate of the *MobiScope* platform. Then the proxy establishes an SSL connection with the intended target, impersonating a mobile device. Using the traffic dumped by the tcpdump process as shown in Figures 1 (c) and 1 (d), and using the private key generated by the squid proxy to communicate with the mobile device, we can decrypt all SSL traffic. We note that when traffic is not encrypted using SSL, the proxy simply acts as a transparent proxy.

This approach will fail for any app that does not trust certificates signed by anything other than a well known root authority. Surprisingly, this is rarely the case. Whereas the Twitter application and the Firefox browser prevent SSL bumping by validating the root certificate, Google Chrome, Safari, the Facebook application, the Google+ application, the default mail clients, and advertisement services do not check the validity of the root certificate. This enables our approach to provide visibility into secure channels established with a wide range of popular apps. We will discuss further this issue in Section **??**.

2.2 Limitations and Deployability

[3]Note that for privacy reasons we do not decrypt traffic generated by human subjects; rather, we use this for controlled experiments in the lab setting.

MobiScope provides a scalable way to achieve pervasive, portable and passive monitoring of network traffic from mobile devices. In this section, we discuss several issues that impact the coverage and deployability of our approach. Note that these limitations have not significantly impacted our ability to measure mobile networking traffic or to deploy our approach to users.

2.2.1 Limitations

[Dave: Add text about monitoring only one interface.]

At most one tunnel. Currently iOS and Android support exactly one VPN connection at a time. This allows *MobiScope* to measure traffic over either WiFi or cellular interfaces, but not both at once. The vast majority of traffic uses only one of these interfaces, and that interface uses the VPN.

Proxy location. All traffic is proxied through a *MobiScope* box, thus the Web services will see the *MobiScope* box address as the end-point and not the mobile device. This might have an impact in case of Web service tailoring the answer according to the IP address of the mobile device (e.g., in case of localization). The biggest problem is when a Web service deny access to some geographic area, but this problem can be worked around by installing a *MobiScope* box on a local (to the Web sevice) machine.

ISP support. Some ISPs block VPN traffic. In that case it is not possible to use the *MobiScope* platform from a mobile device connected to such an ISP. There are few ISPs blocking VPN traffic, and there is a strong incentive to enable VPN traffic in order to attract professional clients.

Limited ISP characterization. *MobiScope* cannot detect traffic differentiation or any other techniques that ISPs use to interpose on network traffic using deep packet inspection (such as advertisement insertion [**TBD: AR: give a reference**]) or optimization (such as traffic compression [**TBD: AR: give a reference**]). This is because the traffic between mobile devices and *MobiScope* is encrypted.

IPv6. *MobiScope* cannot be currently used on networks using IPv6 because IPv6 is not fully supported by mobile devices. Indeed, we observe that though iOS and Android support IPv6 they currently do not support IPv6 traffic through VPN tunnels.

[**TBD: Should we use the tripewire experiment? Currently the description looks like a very small contribution, and it does not bring much to the discussion.**]

2.2.2 Deployability

MobiScope uses standard and often open-source software to manage and record traffic from mobile devices, making it easy to deploy to users and servers. However, a key question is whether the system is sufficiently efficient to minimize its impact on both controlled and in-the-wild experiments. We show empirically that the overheads are reasonable, and provide a brief discussion of incentives for users to adopt our system for in-the-wild experiments.

We identify the following key aspects of user-perceived inefficiency from proxying their network traffic through *MobiScope*:

- **Establishment delay.** We made a simple set of 50 VPN establishments on both iOS (on an iPhone 5 running iOS 6.1) and Android (on a Galaxy Nexus running Android 4.2), and for both Wi-Fi and cellular connections. For Android, we found maximum VPN establishment of 0.81 second on Wi-Fi and of 1.59 second on cellular. For iOS that uses the older IKEv1, the VPN establishment takes longer: we observe a maximum of 2 seconds on Wi-Fi and 2.18 seconds on cellular. In summary, for most long term traffic monitoring experiments, the VPN establishment delay is negligible. [Dave: median? AL: I removed the median to reduce too many references to numbers. In that case, the max is enough to support our argument.]

- **Encapsulation overhead – data consumption.** *MobiScope* uses IPsec for datagram encryption, thus there is an encapsulation overhead for each packet exchanged between the mobile device and the *MobiScope* box. To evaluate this overhead, we logged for 30 days and 25 mobile devices the size of all IPsec packets and of the encapsulated packet. We observe a maximum increase in the packet size due to the IPsec encapsulation of 12.8%. Within the scope of the traffic monitoring experiments performed with *MobiScope*, the impact of this overhead is negligible. However, in case of experiments with a limited cellular data plan, this overhead must be taken into account.

- **Encapsulation overhead – power consumption.** To establish and maintain a VPN tunnel, the mobile devices need additional resources that translate into a larger battery drain during experiments. To evaluate this battery consumption due to the VPN, we used a power meter to measure the draw from a Galaxy Nexus running Android 4.2. We run 10-minute experiments with and without the VPN enabled. For each experiment, we generated an intensive activity such as Web searches, map searches, Facebook interaction, e-mail and video streaming. We found that the VPN leads to a 10% power overhead.

  We used a power meter on an Android device only because power measurements require physical access to the battery for a device, which is not feasible for iOS devices. For iOS devices we conducted an experiment using video streaming to drain a fully charged battery with and without the VPN enabled. We again found approximately 10% power overhead.

  In summary, the power overhead is low enough to run long experiments using mobile devices on *MobiScope*. [**TBD: It seems that NULL encryption is not possible for VPNs on iOS and Android. I don't believe we should claim we are exploring this option if we**

already know it will not work without a dedicated app (which is against the mobiscope design)]To further reduce this overhead (which is primarily for cryptographic operations), we are investigating using NULL encryption (*i.e.*no encryption) for users that do not need the additional privacy enabled by our VPN proxy.

Incentives. *MobiScope* is a fantastic platform for controlled experiments. However, it is also important to measure traffic from real users, so we must provide an incentive for them to use the platform. We have developed a variety of user incentives that can offset the costs of *MobiScope*; here, we name a few of the most interesting ones.[4] For example, we can provide users with fine-grained views of privacy leaks from their applications, and allow them to enable a *MobiScope* plugin that blocks Personally Identifiable Information (PII). In addition, we are investigating the opportunities for page speed optimization. Last, we have implemented opt-in device-wide ad-blocking, which can reduce the volume of costly cellular traffic transferred by a device.

3. DATASETS DESCRIPTION

Using *MobiScope*, we collected two different datasets that we use in the next sections to analyze key characteristics of iOS and to compare them with Android. In the following, we describe these two datasets, one dataset has been collected using controlled experiments, the other dataset has been collectected using IRB-approved in-the-wild measurements during seven months on a small set of real users.

The collection of these two very different datasets shows the flexibility of *MobiScope* to perform a large variety of measurements on the traffic of mobile devices.

3.1 Controlled Experiments

We made all our controlled experiments using three devices: one Galaxy Nexus running Android 4.2, one Google Nexus running Android 4.0, and one iPhone 3GS running iOS 6. We start each set of controlled experiments with a factory reset. Then we connect to device to the *MobiScope* platform, we enable the SSL-Bumping plugin, and we start the experiment.

The first set of controlled experiments consist in manually testing the 100 most popular free Android apps in the *Google Play* store and [**TBD:**] iOS applications from the iOS App store [**TBD: give a date**]. For each application, we install it, enter user credentials for the account if it is relevant, play with it for [**TBD:**] minutes, and uninstall it. This experiment is a characterization of popular applications with real user interactions in a perfectly controlled environment.

The second set of controlled experiments consist in fully-automated experiments on the most popular 908 Android applications from a free, third-party Android market[**TBD: which market, we must give the name**]. We perform this test because Android devices can install *Third-party applications* that are not available on the *Google Play* store. So, it is important to characterize these applications that do not have to follow the Android market publication process[**TBD: Is there different constraints on this free market**]. To automate the experiment process we use the *adb* Android command shell to install each app, connect the device to the *MobiScope* platform, and start the app. Then we use *Monkey* [**TBD: give a ref**], an adb stress tool, to perform a series of 10,000 actions which includes random swipes, touches, and text entries. Finally, we use adb to uninstall the application and reboot the device to forcibly end any lingering connections. This second set of experiment is limited to Android devices because iOS does not provide an equivalent to adb to manage apps installation.

3.2 In The Wild Measurements

The controlled experiments described in Section 3.1 are important to characterize the behavior of applications in a controlled environment. However, it is also important to characterize the behavior of applications when used by real users during several months. For this reason, we performed an IRB-approved measurements during seven months, from October 20, 2012 to May 20, 2013 with a small set of real users.

We deployed two *MobiScope* servers, one in the USA and one in France that were used by 26 devices: 10 iPhones, 4 iPads, 1 iPodTouch, and 11 Android phones. The Android devices in this dataset include the Nexus, Sony, Samsung, and Gsmart brands while the iPhone devices include one iPhone 3GS, four iPhone 5, and five iPhone 4S. These devices belongs to 21 different users, volunteers for our IRB approved study. This dataset, called *mobWild*, consists of 218 days of data monitored on the *MobiScope* servers; the number of days for each user varies from 5 to 215 with a median of 35 days. We note that the SSL-Bumping plugin has been disabled for all experiments involving real users.

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 our lab, where the risks and benefits of our study are clearly explained. The incentive to use VPNs was a lottery of Amazon.com gift certificates. To protect the identity of information leaked in the data, we use public key cryptography to encrypt all the tcpdump outputs; the private key is maintained on separate secure severs and with access limited to approved researchers. Furthermore, users are free to delete their data and disable monitoring at any time. For privacy reasons, we will not make this data publicly available.

4. MOBILE TRAFFIC CLASSIFICATION

MobiScope offers a network perspective of the mobile Internet traffic. For a meaningful analysis of the traffic captured by *MobiScope*, we must be able to identify the access technology used by the devices, and the applications and

[4]We have implemented most of these examples, but their details are beyond the scope of this paper.

IP Protocol	Service	Android		iOS	
		Cell.	Wi-Fi	Cell.	Wi-Fi
TCP	HTTP	35.386	68.686	52.109	75.506
	SSL	61.135	27.366	46.765	18.777
	other	2.346	3.290	0.256	1.818
UDP	DNS	0.682	0.496	0.545	0.305
	other	0.316	0.098	0.286	3.583
Other	-	0.135	0.064	0.039	0.011
total		100.00	100.00	100.00	100.00

Table 1: Traffic volume (in percentage) of popular protocols and services on Android and iOS devices over cellular and Wi-Fi. *TCP flows are responsible for more than 90% of traffic volume. Traffic share of SSL over cellular networks is more than twice the traffic share of SSL over Wi-Fi.*[Dave: Total-¿overall, put fractions of overall traffic for cell/wifi]

Web-services responsible for each flow. In this section we describe the technique we used to identify the access technology and the applications and Web-services and show that *MobiScope* can be used characterize the behavior of mobile applications.

4.1 Access Technology Classification

To quantify the impact of the access technology, Wi-Fi or cellular, we need to first identify the access technology used by the devices to connect to the Internet. We estimate the access technology with the AS description obtained by performing a *WHOIS* lookup on the IP address used by the mobile client. We use information from *whois.cmyru.com* and *utrace.de* WHOIS databases to manually classify the ASes as cellular or Wi-Fi. Based on this classification, the *mobWild* dataset consists of traffic from 54 distinct ASes, of which we classify 9 to be *cellular* ASes. During the measurement study, each device connected our *MobiScope* server from at most two distinct cellular ASes. In contrast, a median of 4 Wi-Fi ASes were observed per device and for one device we observed traffic from 25 different Wi-Fi ASes spread across 5 countries.

This classification technique fails when a device uses a Wi-Fi access-point that internally connects to the Internet using cellular networks. Our technique wrongly classifies flows from such Wi-Fi connections as cellular; a Wi-Fi home gateway of one device in the *mobWild* dataset falls into this category.

4.2 Classification of Mobile Applications and Services

Mobile applications, and OS services and libraries available to these applications, rely on HTTP and SSL to exchange data [14, 10, 23]. For our analysis, we focus on identifying the applications, OS services, and other Web-services responsible for these HTTP and SSL flows.

We begin our identification process using the classification provided Bro [16]. Bro uses the protocol field in the IP header to broadly classify the flows, and we use this classification to label flows as either TCP, UDP, or *other*. Bro further classifies TCP flows using well defined port numbers, and we use this classification to label flows as either HTTP, SSL (which includes HTTPS, IMAP, etc.) or *other* flows. Similarly, we use Bro to label UDP flows as either DNS or *other*. Indeed, in Table 1, we observe that more than 92% of the traffic in our *mobWild* dataset is either HTTP or SSL. We also observe that the share of HTTP volume over Wi-Fi and cellular are significantly different. This increase is a result of the reduced share of media traffic and the use of email and for social networking services that rely on SSL. [**TBD: where can we see this?**]

User-Agent
YahooMobileMail/1.0 (Android Mail; 1.4.6) (crespo;samsung;Nexus S;4.1.2/JZO54K)
AppleCoreMedia/1.0.0.10A523 (iPad; U; CPU OS 6_0_1 like Mac OS X; en_us)
Dalvik/1.6.0 (Linux; U; Android 4.2.2; Nexus 4 Build/JDQ39)

Table 2: *User-Agent* strings. *The first string contains the application identifier, the second hides the application and describes the OS service/library used, while the third does not contain any useful signature.*

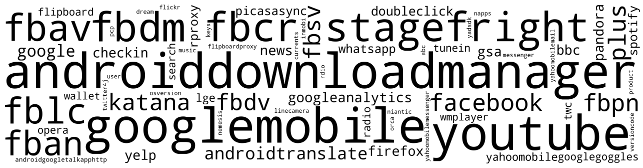
OS	Store	# Apps	Generated HTTP Traffic	Correct User-Agent
iOS	App Store	209	176	149 (84.6%)
Android	Google Play	100	92	21 (22.8%)
Android	Third party	908		

Table 3: Classification of applications based on *User-Agent*. *We observe that we were table to classify [***TBD:***] of iOS and [***TBD:***] of Android applications that generated traffic during our experiments.*

4.2.1 HTTP Traffic Classification

HTTP is used by Mobile applications to fetch data from Web-services. Indeed, the *Host* field in the HTTP header and the IP addresses to which the device contacts over HTTP can be used to identify the Web-services. However, such classification based on *Host* field and IP address shall hide the application used to contact the Web-service. For example, popular Web-services such as Facebook and Twitter can be accessed either through a web-browser or through dedicated their mobile applications. To identify flows from their dedicated mobile applications, Web-services are known to rely on the *User-Agent* field. Indeed, *User-Agent* based classification has been used to isolate mobile traffic, however, rather than identifying individual applications, these studies limited their classification granularity to the category of the application [9, 23, 14]. We now use the results from our controlled experiments and the *mobWild* dataset to argue that the *User-Agent* can be used to identify flow from popular iOS and Android applications, for flows that do not contain a useful *User-Agent* we fall back to classifying using the *Host* field.

(a) iOS



(b) Android

Figure 3: *User-Agent* signatures in iOS and Android HTTP flows. *The font weight represents the number of users for which a particular signature was observed.*

In our controlled experiments we observed a non-empty *User-Agent* string in more than [**TBD: 99.7**]% of the HTTP flows from iOS and [**TBD: 90.9**]%flows from Android. A *User-Agent* strings may contain an application identifier and other auxiliary information such as details of the OS, manufacturer, and compatibility with Web-browsers [1]. For example, the first *User-Agent* in Table 2 contains the information of the application, YahooMobileMail, while the second *User-Agent* hides the application and specifies the AppleCoreMedia service of iOS, and the third does not provide any useful information. We use a set of regular expressions to filter out the auxiliary information and further cluster these extracted tokens using the edit distance between the tokens[5].

We now use Table 3 to summarize the usefulness of *User-Agent* based classification for the traffic observed during our experiments. While testing 209 iOS applications we observed HTTP traffic from 176 applications; 149 of these were correctly identified based on the signatures in their *User-Agent*, which we verified by manual inspection. For the 27 applications which we failed to identify, we observed signatures for OS services and libraries, such as applecoremedia, gamekit, geoservices, etc. and signatures of third-party libraries and services such as *Google Analytics* and *Adobe Air*. For example, while testing streaming media applications such as YouTube we observed a signature of *applecoremedia*.

While testing 100 Android applications we observed HTTP traffic from 92 applications; 21 of these were correctly identified based on the signatures in their *User-Agent*, which we verified by manual inspection. The rest of the traffic used *User-Agent* without any application signatures, or signatures of libraries such as *StageFright* and *Adsense for mobile*. However, the popular applications such as Facebook, Spotify, Twitter,

[5]We plan to release this code along with *MobiScope* package.

Category	iOS		Android	
	Bytes	**Flows**	**Bytes**	**Flows**
Media (Popular)	51.405	12.131	65.922	22.377
Application	33.987	80.758	31.353	77.498
Ads and Analytics	[**TBD:**]	[**TBD:**]	[**TBD:**]	[**TBD:**]
Media (Other)	14.572	5.914	2.712	0.044
Other	0.036	1.1963	0.013	0.081
total	100	100	100	100

Table 4: Classification of HTTP Traffic. *A total of [**TBD:**] iOS and [**TBD:**]Android applications were responsible for 80% of iOS and 77% of Android HTTP flows.*

On using this technique on the *mobWild* dataset we were able to identify [**TBD:**] applications, which we summarize in Figure 3. The *word cloud* in Figure 3 contains the signatures we were able to extract from *User-Agent* field; the text size of the signature represents the number of users for which the signature was observed. Along with signatures of applications such as iTunes and YouTube, we also observe signatures of the *applecoremedia* and *stagefright* services that are responsible to download media content on iOS and Android devices respectively. Across the iOS and Android devices, we observe a total of [**TBD: 1435**] unique *User-Agent* strings that produce [**TBD: 361**] unique signatures which we cluster to [**TBD:**] different applications and services.

The use of OS services and media libraries to fetch media content resulted in the signature of AppleCoreMedia in more than 98.45% of the content downloaded from the YouTube servers (which we identify based on the *Host* field in the *HTTP GET* requests). Similarly, depending the Android version we observe either the signature for Stagefright[2] or no application or OS service signature for YouTube traffic to Android devices depending on the OS version. For flows that used AppleCoreMedia, we observed signatures for popular media services such as Netflix, YouTube, Vimeo, Pandora, etc. in the *Host* field in the majority traffic.

In Table 4, we observe that with a combination of *User-Agent* and *Host* field in HTTP headers, we were able to classify more than 98% of the traffic in terms of flows and bytes from iOS and Android devices. We observe that media from popular hosts contribute to more than 50% of the traffic volume from iOS and Android devices. Similarly, we were able to identify applications for more than 77% of flows from Android and iOS devices. However, we observe media (identified based on the *User-Agent*) served from CDNs and others hosts from which we could not identify the webservice from other fields in the HTTP header and the DNS responses before the HTTP flows to be 14.5% of the traffic volume for the iOS devices in our dataset.

4.2.2 *Classification of SSL Traffic.*

Unlike HTTP flows, SSL flows provide limited information that can be used to identify the applications. Our objective classifying SSL traffic was therefore focused towards identifying the Web-services responsible for the SSL flows.

Service	iOS		Android	
	Bytes	**Flows**	**Bytes**	**Flows**
HTTPS	91.287	81.960	97.852	97.168
Mail	6.700	15.872	0.689	0.320
Notification	1.412	1.553	1.321	2.100
Other	0.601	0.615	0.138	0.412
total	100	100	100	100

Table 5: Classification of SSL Traffic based on port number. *HTTPs is the most popular service that uses SSL in the* mobWild *dataset.*

We now show how we used the port number, the SSL certificate with server name identification, and DNS queries to identify the source of SSL traffic.

Mobile devices use SSL for various services including mail, notifications, instant messaging, and web browsing. Services such as mail, instant messaging, and notifications are documented to use dedicated port numbers of their traffic[6] On using port numbers, we observe in Table 5 that a majority of SSL traffic by volume and flows is HTTPS. We then focus our attention on indentifying the Web-services responsible for the HTTPs flows.

We first use the common name (CN) field of certificates to identify the servers that exchanged data using HTTPS. We observe that less than 25% of the HTTPS traffic from iOS and Android contains the fully qualified domain name (FQDN) in the subject of the certificate; the rest of the traffic either contains regular expressions such as *.google.com in the certificate or is a continuation of a previous SSL session. To further resolve the hostnames, we rely on *server name indication* used by SSL flows [6]. Servers that host multiple services use the *server name indication* to distinguish these services. For example, we observe a *server name indication* of *plus.google.com* and *s.youtube.com* in two flows that used a certificate with a CN **.google.com*. However, we observe that by using either the certificate or the *server name* we were able to identify the name of the Web-service in less than [**TBD: 40**]% of iOS and Android HTTPS traffic.

For such flows we use DNS requests made by the mobile devices before starting the HTTPS flows, a technique similar to DN-Hunter [4]. DN-Hunter relies on the most recent FQDN that corresponds to the IP address, however in our controlled experiments we observe Android and iOS devices use the first entry in DNS response while resolving *hostnames*. We therefore use the latest DNS response that contains the IP address of the webservice in the first position of the DNS response. Indeed, for 97.8% of the Android and 83.4% of the iOS HTTPS traffic that we could not classify using other fields, we observe that the latest DNS response before the flow started contained the IP address of the web-service as the first entry in the DNS response[7]. Despite the potential usefulness of DNS responses, we give a high priority to the server-name and the certificates because we observed that for flows that contained the server name did not contain the same name in the DNS response for 9.2% of the iOS traffic and 5.6% of Android traffic.

[6]We also use the AS for identifying the notification messages as detailed in Section **??**.

[7]The share of SSL traffic where the latest DNS response contains the IP address of the web-service in the first position is 97.4% for Android and 88.6% of iOS

iOS	Android
imap.gmail.com	picasaweb.google.com
www.google.com	www.googleapis.com
sphotos-a.xx.fbcdn.net	android.clients.google.com
itunes.apple.com	clients4.google.com
m.google.com	fbcdn-photos-a.akamaihd.net

Table 6: Popular hostnames for iOS and Android based on traffic volume.

Service	iOS		Android	
	Bytes	**Flows**	**Bytes**	**Flows**
Mail	9.970	62.168	1.626	1.565
Social Networking	12.491	6.683	36.661	22.352
Apple/Google Store	5.457	3.463	0.044	0.036
Instant Messages	0.982	7.089	1.411	3.109
Other Google Services	58.665	13.32510	45.024	46.089
total	87.564	92.728	84.776	73.151

Table 7: Sample classification of SSL traffic based on names in certificate, server name identification, and DNS request.

In Table 6 we present the top5 hostnames that were responsible for 66% of iOS traffic and 54% of Android by volume. We observe that despite the hostname we cannot uniquely identify the application. For example, *www.googleapis.com* and *clients4.google.com* offer limited information on the application or web-service that is responsible for the content; one can only guess that these flows belong to some google service. However, hostname such as *fbcdn-photos-a.akamaihd.net* gives an indication that the traffic is due to facebook.

In Table 7 we show how we grouped SSL traffic based on names identified using the SSL fields and DNS queries and port numbers. We observe that the iOS devices in our dataset generated a significant number of flows to email sites. Similarly, we were able to group 12.49% of iOS and 36.661% of Android traffic with social network services that includes *Google Plus, Facebook, and Twitter*. We speculate the increase in traffic share for Android devices is because Android devices offer services to backup photos on *Google Plus*. Similarly, we observe that 5.4% of the traffic from iOS devices was from Apple stores while we observe only 0.04% of traffic to the *Google Play* store. This low share is because google can use hosts matching the pattern *client*.google.com* to serve for different Web-services. We observed a similar behavior in our controlled experiments, and we group such traffic as *other google services*. Indeed, in Table 7 we observe that traffic from Google is the largest source of SSL traffic for iOS and Android devices in our dataset.

4.3 Case Study: Facebook Application in the Wild

In summary, we use *MobiScope* to perform controlled experiments and in the wild measurements to characterize mobile Internet traffic. We use Bro to analyze the data and build on the output of bro to further classify HTTP flows and SSL flows to identify the source of the traffic. We now present the results of our experiments and measurements study.

5. PRIVACY INVASIVE SERVICES

[**TBD: to revise**] We now use pervasive nature of *MobiScope* to detail the privacy invasiveness of applications and Web-services. For our analysis we concentrated on the data sent from the mobile devices with a focus on the *what* data is sent, *to whom* is the data sent, and *how frequently* is data sent.

5.1 Personally Identifiable Information Leaks

For our experiments, we created fake user accounts with fake contact information, and fake Twitter and Facebook accounts. Our goal is to detect if any Personally Identifiable Information (PII)—email address, phone number, IMEI number—stored on the device is leaked across the network over HTTP or HTTPS (using the SSL bumping plugin). While we acknowledge that some of this information may be relevant for the application, we strongly believe that this information should never travel across the network in plaintext (HTTP).

In Table 8, we present the different personally identifiable information leaked for both Android and iPhone apps. We observe that the IMEI, a unique identifier tied to a phone, is the most commonly leaked PII for Android applications. This IMEI can be used to track and correlate a user's behavior across Web-services. Similarly, we observe that Android applications leak the Android ID, a unique identifier tied to an Android device. In Table 8, we also observe that other information like contacts, emails, and passwords are leaked in the clear. The email address, the address used to sign up for the services, was leaked in the clear by 13 iOS and 3 Android applications from our set of popular applications.

During our experiments, we observed that personally identifiable information information is also sent over HTTPS. In the following, we focus on device identifiers such as the IMEI and the Android device ID. In Table 9, we present the top 10 sites ordered by the number of flows that sent the IMEI over HTTPS. We observe that four of the top 10 sites that receive this information are ads and analytics sites.

Finding ads and analytics sites that receive personally identifiable information from apps is an abuse of the permission the user is giving to the apps. Indeed, if an app ask to access device identifiers and the user grant this access, the user is never notified that third parties will have access to these identifiers. Out of the 77 sites that received either the IMEI or Device ID in the clear or over HTTPS, 35 sites were third party ads and analytics sites.

Host	IMEI	Device ID	*Ads & Analytics*
chartboost.com	✓	✓	✓
tapjoyads.com	✓	-	✓
getjar.com	✓	✓	-
pocketchange.com	✓	✓	-
iheart.com	✓	✓	-
aarki.net	✓	-	✓
zynga.com	✓	-	-
droidsecurity.appspot.com	✓	-	-
google.com	-	✓	-
flurry.com	-	✓	✓
groupon.com	-	✓	-

Table 9: Top 10 hosts that receive the IMEI or Device ID over HTTPS. *Hosts are ordered by the number of flows that send the IMEI number, followed by the number of flows that send the device ID over HTTPS. Four of the top 10 hosts that receive this information are ads and analytics sites.*

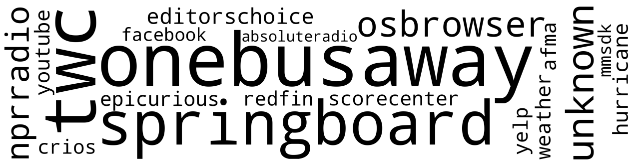


Figure 4: Applications that send the location information in the clear. *The font size represent the number of flows that sent the location information in the clear.*

In summary, we use our controlled experiments to identify PII leaks on both HTTP and HTTPS, and we show that PII are leaked to third party sites such as ads and analytics. These controlled experiments are a practical use case of *MobiScope*, experiments requiring warranty voiding the devices otherwise. In particular, *MobiScope* enable to reveal PII leaks over HTTPS.

5.2 PII in the Wild

In the previous section, we focused on controlled experiments. We now focus on the analysis of the *mobWild* dataset. For this dataset, for evident ethical reasons, we did not run the SSL bumping plugin because it would have revealed credentials of most of the applications used by our pool of real users. Instead, we focus on which information is leaked in the clear.

In Figure 4, we present a *word cloud* of the applications that send the location information of the devices. We observe that a bus service application (*One Bus Away*), the application that manages the iOS homescreen (*springboard*), and weather applications (*twc*, *weather*, *hurricane*) were responsible for more 78% of the flows that sent the location information in the clear. Moreover, we observe that 4% of the flows sent the location information to ads and analytics sites; more than 80% of *ad-flows* leaking location information did not include an application signature in the user-agent field, the rest of the flows being from apps including browsers, the Facebook app, and angry birds.

In addition, we also observe that the device ID and IMEI

Store	Platform	# Apps	Email	Location	Name	Password	Device ID	Contacts	IMEI
App Store	iPhone	209	13 (6.2%)	20 (9.5%)	4 (1.9%)	0 (0%)	16 (7.6%)	0 (0%)	0 (0%)
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%)

Table 8: Summary of personally identifiable information leaked in plaintext (HTTP) by Android and iPhone applications. *The popular iOS applications tend to leak the location information in the clear while Android applications leak the IMEI number and Android ID in the clear.*

Tracker	Number of devices tracked		
	Total	iOS	Android
doubleclick.net	26 *(all)*	15 *(all)*	11 *(all)*
google-analytics.com	26 *(all)*	15 *(all)*	11 *(all)*
googlesyndication.com	22	12	10
admob.com	21	11	10
scorecardresearch.com	21	11	10

Table 10: The top 5 ads and analytics sites that were contacted by the devices in our dataset. *The sites, doubleclick.net and google-analytics.com, were contacted by all the 26 devices in mobWild.*

number are leaked in the clear in the *mobWild* dataset. Based on our classification methodology, we observe that the IMEI number and device ID is leaked by the Web-browser; we do not observe any application signature in non-browser flows in the *mobWild* dataset that leaked the IMEI number or the device ID in the clear. As in the case of controlled experiments, ads and analytics sites are the most popular destination for the IMEI number leaks. Among the 16 sites that sent the IMEI number in the clear, 10 sites are ads and analytics sites; the rest of the sites includes sites for games, news, and manufacturer updates.

[**TBD: focusing on ads and analytics is just what you did in the previous paragraph. What is new or different in this one.**] We now focus our attention on the extent to which devices in the *mobWild* dataset contact ads and analytics (A&A) sites, an activity that is receiving considerable attention [18, 13, 21]. Using our classification based on the *Host*, we observe that the ads and analytics traffic was responsible for up to 6% of the traffic by volume per device, an observation in line with the one made by Vallina-Rodriguez *et al.* [21]. Rather that focusing on the traffic volume we focus on the extent to which these sites are able to track the users in the dataset and the applications that facilitate this tracking.

In Table 10 we present the number of A&A sites ordered according to the number of devices that contacted them. We observe that all the devices in the *mobWild* dataset contacted doubleclick.com, an ad site, and google-analytics.com, a tracking site. Furthermore, we observe that 66.12% of the volume of ad traffic in the *mobWild* dataset was from the browsers, 6.46% of the traffic contained a blank user-agent field, and 4.8% of the traffic contained a signature of *Google-Analytics*[8].

[8]This signature was observed even in the flows for users that did not have the Google Analytics application installed on the device.

The rest of the traffic contained signatures of other applications such as Facebook, Pandora, and YouTube.

[**TBD: Add some comment about ConVis (Nick's work), say that it's another contribution in that it helps average users visualize their tracking and PII, and that we make it available to participating users, and the URL for the demo is at location X.**]

6. MISC

7. RELATED WORK

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 [15], network bandwidth and leak private information [8, 12]. Similarly, AppInsight [17] and PiOS [7] 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.* [21] uses this approach to characterize performance and the impact of advertising. Gerber *et al.* [11] similarly use this approach to estimate network performance for mobile devices. [14] [5] 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 [22, 19] 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.

	Network Coverage	Portability	Deployment model	Meas. Type
AT&T/Telefonica study [21, 11]	Single carrier	All OSes	Instrument cell infrastructure	Passive
WiFi study [5]	Single WiFi network	All OSes	Instrument WiFi network	Passive
PhoneLab/TaintDroid [8]	Multiple networks	Android	Install custom OS	Active/Passive
MobiPerf [22]/SpeedTest [19]	Multiple networks	Android	Install App	Active
MobiScope	Any network	Android / iOS	VPN configuration	Passive

Table 11: 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.

8. CONCLUSION

Placeholder

9. REFERENCES

[1] Browser detection using the user agent. `https://developer.mozilla.org/en-US/docs/Browser_detection_using_the_user_agent`.

[2] Media — Android Developers. `http://source.android.com/devices/media.html`.

[3] Open vswitch: An open virtual switch. `http://openvswitch.org/`.

[4] I. N. Bermudez, M. Mellia, M. M. Munafo, R. Keralapura, and A. Nucci. DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proc. of IMC*, pages 413–426, New York, NY, USA, 2012. ACM.

[5] 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.

[6] D. Eastlake et al. Rfc 6066: Transport layer security (tls) extensions: Extension definitions, 2011.

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

[8] 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.

[9] J. Erman, A. Gerber, and S. Sen. HTTP in the Home: It is not just about PCs. *ACM SIGCOMM Computer Communication Review*, 41(1):90–95, 2011.

[10] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. *Proc. of IMC*, page 281, 2010.

[11] 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.

[12] 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.

[13] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo. Don't kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market. In *Proc. of Hotmobile*. ACM, 2012.

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

[15] 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.

[16] V. Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

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

[18] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. *Proc. of USENIX NSDI*, 2012.

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

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

[21] 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.

[22] 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.

[23] 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.