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 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 6-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 lease one of the above dimensions of mobile Internet traffic [**?**, **?**, **?**, **?**], 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 the *MobiScope* plateform[1] 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 experimental platform.
6. *Single machine.* We want *MobiScope* to fit on a single machine to facilitate its installation and deployment.

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

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

2.1 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.

2.1.1 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* [11], 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 [**?**] for authentication and key negotiation.

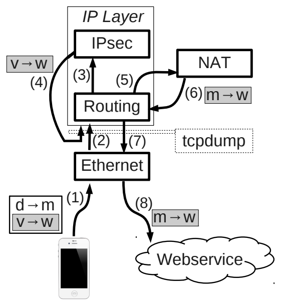
2.1.2 Monitoring Traffic on MobiScope

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 Web service. 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*.

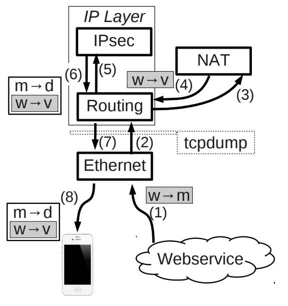
Figure 1 (a) shows the processing on *MobiScope* of packets sent from a mobile device to the Internet. At step (1), (2), and (3) the encrypted datagram (in gray) goes up the layers, this datagram is encapsulated in a regular IP datagram sent from the device (address *d*) to the *MobiScope* box (address *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 Web service (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 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 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 Web service to the *MobiScope* 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.

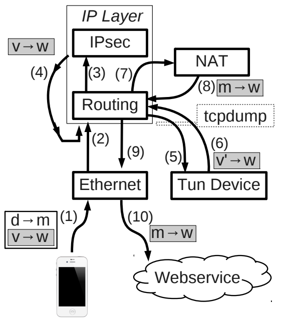
Our solution is to force the packets from *w* to *v* to make a loop on a TUN device, and to dump these packets at the TUN device, see Figure 1 (d). Indeed, when a packet is received from the Internet is goes up the layers up to the NAT at step (3). At that point, the NAT resolve the public address of 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



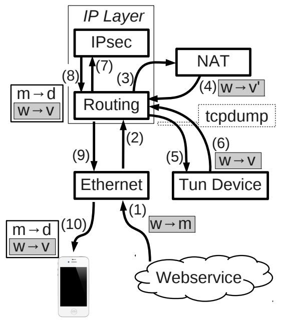
(a) Packet from mobile device. *Tcpdump can capture packets at step (2) d → m, (4) v → w, and (7) m → w.*



(b) Packet to mobile device. *Tcpdump can capture packets at step (2) w → m and (7) m → d, however it is 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.
$\boxed{a \rightarrow b}$	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.

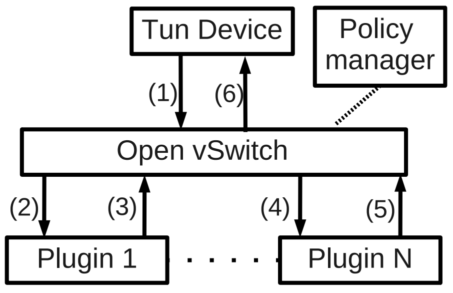


Figure 2: Plugin Infrastructure on *MobiScope*.

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.

2.1.3 *Modifying Traffic on MobiScope*

We described in section 2.1.2, 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 the plugin infrastructure of *MobiScope*. When a packet is received at the TUN device, it is sent to an open vSwitch [5] process whose goal is to define in which order each plugin will receive packets. This order is configured by the policy manager. So packets flow through each plugin, being processed at each plugin.

Plugins can be used for many different purposes such as replaying traffic, testing reliability to data corruption or delays, or decrypting SSH traffic. In the following, we describe how easy it is to perform SSH traffic decryption using the *MobiScope* plugin infrastructure.

In order to perform SSH traffic decryption, we use a feature of the Squid proxy called SSL bumping[**TBD: AR: give a reference**]. This feature consists in performing a man-in-the-middle attack. When the mobile device connects to a Web service, the squid proxy impersonates the Web service using a forged certificate signed with the root certificate of the *MobiScope* platform (see Section 2.1.1). Then the squid proxy negotiate with the Web service a SSL session, 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 notice that when traffic is not encrypted using SSL, the squid proxy simply acts as a transparent proxy.

Impersonating a Web service using a root certificate that is not issued by a well known root authority might fail if the application on the mobile device only allows certificate issued by well known authorities. 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, thus the SSL bumping attack is possible. We will discuss further this issue in Section **??**.

2.2 Limitations

Using *MobiScope* to monitor traffic from mobile devices leads to limitations that we discuss in the following. We discuss first the overheads due to the VPN on the monitoring: VPN establishment delay, data consumption overhead due to the VPN encapsulation, power consumption overhead on the mobile device. Then we discuss additional limitations such as the one due to the encryption of all the traffic between the mobile device and the *MobiScope* box, encryption preventing the access ISP to perform traffic modification and optimization.

2.2.1 *VPN Establishment Delay*

To establish a VPN tunnel, the mobile device and the *MobiScope* box must negotiate, which takes time. iOS devices use IKEv1 to establish VPN tunnels, whereas Android devices use the more recent and faster IKEv2.

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.

2.2.2 *Data Consumption Overhead*

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.

2.2.3 *Power Consumption Overhead*

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 minutes 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 made simpler experiments consisting in draining a fully charged battery with a video streaming with a without the VPN enabled. We found also around a 10% power overhead.

In summary, the power overhead is low enough to run long experiments using mobile devices on *MobiScope*.

2.2.4 *Limitation*

The design of *MobiScope* comes with a few additional limitations that we discuss in the following.

First, 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.

Second, some ISPs block VPN traffic. In that case it is not possible to use the *MobiScope* plateform 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.

Third, *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.

Finally, because all the traffic between mobile devices and *MobiScope* is encrypted, we cannot observe any traffic modification (such as advertisement insertion [**TBD: AR: give a reference**]) or optimization (such as traffic compression [**TBD: AR: give a reference**]) made by ISPs. Indeed, ISPs that modify traffic must perform deep packet inspection, which is not possible when the traffic is encrypted. Whereas *MobiScope* can be used to explore specificities of ISPs, it has been primarily designed to explore specificities of mobile devices and access technology.

In summary, despite these limitations, we believe that *MobiScope* is a powerful and flexible plateform to monitor the mobile Internet traffic.

[**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.**]

3. METHODOLOGY AND DATASET

We used *MobiScope* to characterize mobile Internet traffic, and detail the impact of access technology and operating systems on application behavior.

Our analysis methodology included controlled experiments to detail the behavior of specific applications and OS services, and a 7-month long IRB approved measurement study to characterize mobile Internet traffic in the *in the wild* .

3.1 Controlled Experiments

For our controlled experiments, we ran the latest versions of Android (Ice Cream Sandwich 4.0, and Jelly-bean 4.2) and iOS 6 respectively on our Android and iOS devices. We analyze the behavior of OS services and the default applications by first performing a factory reset on these devices, and installing the *MobiScope* credentials on this device. We then test Android and iOS applications by installing the application, interacting with the application for a few minutes, and finally uninstalling the application. During our controlled experiment we use SSL-Bumping to study the behavior of SSL traffic from these applications.

Our first experiment included manual testing of the top 100 most popular free Android apps from the *Google Play* store and [**TBD:**] iOS applications from the iOS App store. For this experiment we first manually installed each application by hand, enter user credentials for accounts like Facebook and Twitter, and toy with the app for [**TBD:**] minutes. In addition to this manual setup, we used an automatic test-click generator to further toy with the Android applications for [**TBD:**] actions. We did not perform this automation step while testing the the iOS applications. We then uninstall the application and reset the device to test the next application.

For our second experiment we performed fully-automated tests on 1003 Android applications from a free, third-party Android market. We perform this test because Android devices can install *Third-party applications* that are not available on the *Google Play* store. A consequence of this freedom is that numerous third-party app markets are available on the web whose applications have not received research attention. Our automation used the adb Android command shell to install each app, enable *MobiScope*, and start the app. The system then used Monkey, an adb stress tool, to perform a series of 10,000 actions. These actions included random swipes, touches, and text entries. We then used adb to uninstall the application and reboot the device to forcibly end any lingering connections.

The results of the controlled experiments can be found in Section **??**.

3.2 In The Wild Measurements

Along with controlled experiments we also conducted a measurement study to characterize the mobile Internet in the wild. We now present the description of the dataset and the methodology we used to classify the traffic in this dataset.

3.2.1 Dataset Description

For this study, we deployed two *MobiScope* servers, one in USA and one in France, to proxy Internet traffic from 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 iPhones include one iPhone 3gs, five iPhone 5, and five iPhone 4S. These devices belonged to 21 users, volunteers for our IRB approved study. This dataset, called *mobWild*, consists of 218 days of data that flowed through our *MobiScope* servers; the number days for each user varies from 5 to 215 with a median of 35 days. We would like to point out that though we performed SSL-Bumping during our controlled experiments, we did not perform SSL-Bumping for the traffic in this dataset.

3.2.2 Ethical Issues

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

3.2.3 Identification of Access Technology

A mobile devices can tunnel the traffic through our *MobiScope* servers using either Wi-Fi or cellular networks. We estimate the access technology with the description of the AS through which the mobile client connects to our *MobiScope* server. We get this AS description by performing a *WHOIS* lookup on the IP address used by the mobile client to tunnel Internet traffic. For our analysis, we use the WHOIS databases available at *whois.cmyru.com* and *utrace.de*. We use the information from these *WHOIS* databases to manually classify the ASes to be either cellular or Wi-Fi. Our dataset consists of data traffic from 54 distinct ASes, of which we classify 9 to be belong to cellular networks. The devices connected to our system from at most two 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 that are spread across 5 countries.

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.*

3.2.4 Traffic Summary

We used Bro [9] to analyze the traffic the passed through our *MobiScope* servers. In Table 1 we summarize *mobWild* based on the classification performed using Bro [9]. Bro classifies IP flows using the protocol field in the IP header. We use this classification to label flows as either TCP, UDP, or *other*; flows that are neither TCP nor UDP are classified as *other*. Bro further uses the well defined port numbers to identify the services that use TCP. We use this classification to label flows as either HTTP, SSL (which includes HTTPS, IMAP, etc.) or *other* flows; TCP flows that are not classified as either HTTP or SSL are classified as *other*. In Table 1, we observe that more than 90% of the traffic in our dataset is either HTTP or SSL. We also observe that the share of HTTP volume over Wi-Fi and cellular are significantly different. As detailed in Section **??**, this difference is primarily due to the use of Wi-Fi to transfer media content. We also observe the share of SSL traffic over cellular networks is considerably larger compared to Wi-Fi networks. This increase is a result of the reduced share of media traffic and the use of email and for social networking applications that rely on SSL. We detail the HTTP and SSL traffic from iOS and Android devices in Section **??**

We focus our application classification on TCP because TCP is responsible for than 90% of the traffic volume in our dataset (see Table 1). Furthermore more than 95% of the TCP traffic is due to SSL and HTTP. We therefore focus our attention on HTTP and SSL traffic.

3.2.5 Classification of HTTP Traffic

The HTTP headers for HTTP Request, HTTP Response, and HTTP Entity contain a wealth of information including *User-Agent*, *Referrer*, *Content-Type*, and *Content-Encoding* that can be used to classify HTTP traffic [7]. Recent studies on mobile traffic classification have relied heavily on the *User-Agent* field to classify mobile HTTP traffic [10, 8, 12]. We quantify the usefulness of the *User-Agent* field and show

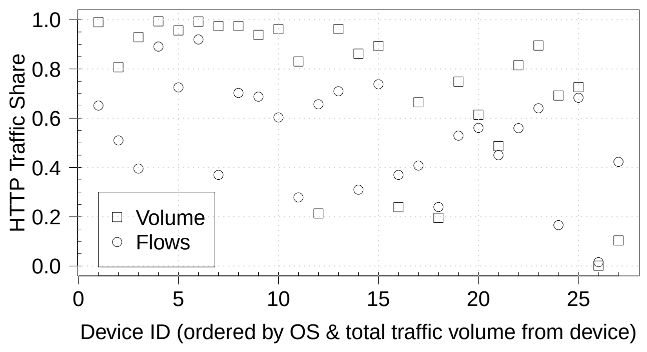


Figure 3: HTTP traffic with a *User-Agent* field containing an identifier of an application (other than Web-browser) or an OS service. *A smaller share of Android HTTP traffic can be classified using User-Agents because Android applications are not limited to underlying OS media services such as AppleCoreMedia.*

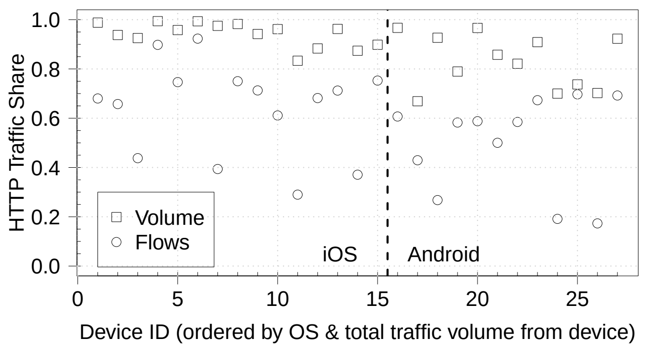


Figure 4: HTTP traffic classified using *User-Agent* or *Host* field provided by popular media services in the HTTP header. *The rest of the traffic is either from Web-browsers and flows that do not include any application signatures in the User-Agent field.*

how other fields such as *Host* are essential to characterize mobile HTTP traffic.

Webservices are known to use the *User-Agent* field to distinguish flows from their mobile applications from the flows originating from Web-browsers. We observe that more than 98% of HTTP traffic from Android and iOS devices in the *mobWild* dataset have a valid *User-Agent* string; we observe a total of 1435 unique *User-Agent* strings across Android and iOS devices. In these *User-Agent* strings we observe that along with the application identifier, the *User-Agent* strings also contain details of the OS, manufacturer, display resolutions, carrier, and other information such as versions and compatibility with other browser engines. We use regular expression to extract the tokens containing the application information and we then cluster these tokens using edit distance between individual tokens and number of matching tokens. We plan to release this code along with *MobiScope* package. 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 3 we plot the fraction of HTTP traffic for we were able to identify an application signature; 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 devices contain an application or OS service signature in the *User-Agent* field, only 23.9% and 19.5% from Android devices with id 16 and 18 contained useful signatures in the *User-Agent* field. On further inspection we observe that this difference is because of the techniques used by Android and iOS application to download audio and video content.

We observe that the iOS devices primarily download media content using the AppleCoreMedia service [1]. We therefore observed a signature for AppleCoreMedia in the *User-Agent* string for 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). We also observed AppleCoreMedia signatures in the content fetched from other media sites such as Netflix and iTunes. Though Android provides a similar service, Stagefright[4], we observe that only 41.91% of YouTube content in our dataset was fetched using Stagefright in the *User-Agent* field. Similarly, we observed content from other media sites were fetched without any unique signatures in the *User-Agent* field. Therefore, because HTTP traffic for Android devices cannot be classified solely based on the *User-Agent* field, and most of the unclassified traffic was media content, we use the *Host* field to further classify flows that do not contain a signature in the *User-Agent* field. [**TBD: Results to show when Stagefright is used and when it is not based on controlled experiments performed**]

In Figure 4 we present the HTTP traffic share that we could classify using the *User-Agent* and *Host* field in the HTTP headers. The rest of the flows belong to traffic from Web-browsers and those that do not include any signatures in the *User-Agent* field.

3.2.6 Classification of SSL Traffic.

Unlike HTTP flows, SSL flows provide limited information that can be used to identify the applications. We now show how we used the *Server-Name* and the DNS queries to classify SSL traffic.

In Figure 5 we observe that relying on the *Server-Name* is not sufficient to classify the traffic. We observe a huge disparity in the fraction of traffic that can be classified using this technique. Along with the *Server-Name*, the common name field of the certificate can be used to identify the traffic. However, the use of CDNs and the use of regular expression

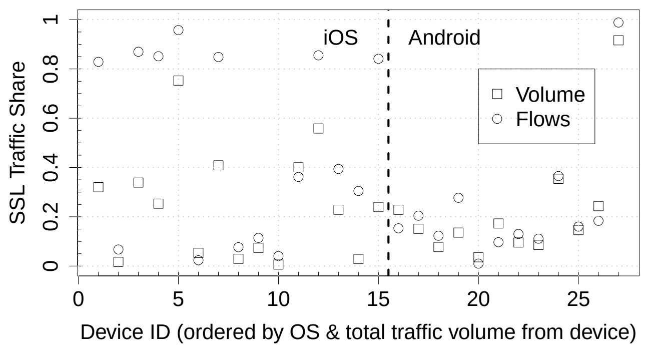


Figure 5: SSL flows classified using the *Server-Name* in the flows.

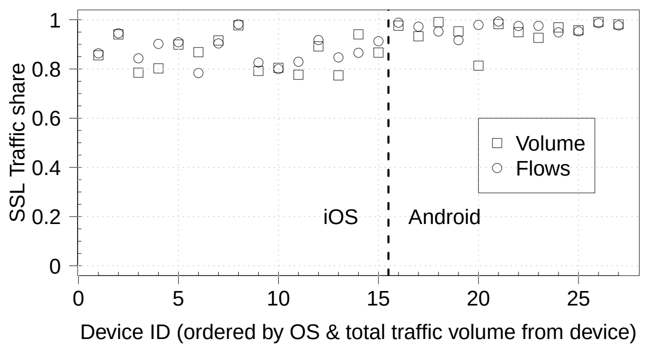


Figure 6: SSL traffic share where the most recent DNS response contained the IP address of the SSL flow in the first position.

to support a large set of hostnames gives a us a result similar to that observed in Figure 5.

[**TBD: The plot can be removed and the discussion with CN field can be merged to indicate relying on certificates and server-names on their own is not sufficient**]

We use the DNS flows that passed through *MobiScope* to further classify the SSL flows, a technique similar to DN-Hunter [6]. 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 prefer the the first entry in DNS response while resolving *hostnames*. Popular webservices such as google are known to use the same pool of IP addresses for various applications, for example the IP for gmail may also be used for search. In Figure 6 we present the fraction of SSL traffic where the most recent DNS response contained the IP address of the SSL flow in the first position. We observe that for the majority of SSL traffic by volume and flows can be classified by using the DNS responses.

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.

4. NETWORK CHARACTERISTICS OF NOTIFICATION SERVICES

Mobile operating systems provide OS level notification services to optimize network usage. The Apple Push Notification service (APNs) and Google Cloud Messaging (GCM) [2] are used by iOS and Android applications respectively to receive notifications from the Internet. In this section we show how *MobiScope* was used to compare the notification services of iOS and Android, and detailing it behavior in the wild.

4.1 Controlled Experiment on Factory Reset Devices

We first detail the detail the behavior of notification services by performing a controlled experiment on *factory-reset* devices. The objective of this experiment was to analyze notification services for devices that are used *out of the box*, and detail the impact of device manufacturer, and pre-installed applications.

For our experiment, we performed a *factory reset* on an iPod Touch, an iPad, an iPhone, a Samsung Galaxy SIII, and a Google Nexus S Phone; the reset was performed after their batteries were fully charged. We then perform the initialization step and assigned a dummy email account as the primary account to each of these devices. We then allowed these devices to connect to the Internet over Wi-Fi through *MobiScope* and monitored the Internet traffic from these devices. We then studied the impact of access technology by letting the iPhone and Samsung Galaxy SIII tunnel traffic through *MobiScope* using cellular networks.

We observed that the traffic volume during the 24 hour periods varied from 19 KB to 97 KB depending on the devices. We classify APNS and GCM messages using the TCP port numbers mentioned in their specifications [2, 3]. Because we did not run any applications in the foreground, notification messages were responsible for largest fraction of the traffic volume; the share was 35% for Nexus S, 88% for the Samsung SIII and around 50% for each of the iOS devices. The share of DNS traffic varied from 10% to 40% for each of the devices while the other services contributed to less than 10% of the traffic volume. The only exception was the Samsumg Galaxy SIII which used location services that contributed to 26% of 47 KB traffic volume generated by the device.

In Figure 7 plot the time between successive messages sent by the notification servers on the ports assigned for notifications. We observe that the inter-arrival time between notifications for the Android devices is at least 900 seconds for more than 80% of the notifications observed. The distribution of the inter-arrival time also depends on the access technology for the Samsung Galaxy SIII phone; we observe steps in the distribution for the SIII phone while we do not observe these steps when the same phone uses Wi-Fi. We do

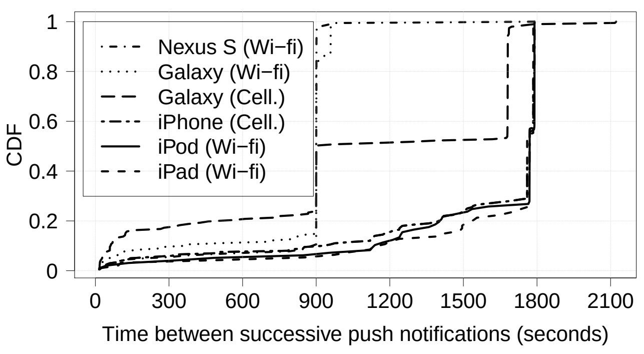


Figure 7: Inter-arrival time between notification messages after factory reset. *The Android and iOS devices communicate with the notification server approximately once every 900 seconds and 1800 seconds respectively. The behavior of Android devices depends on the device, the pre-installed applications, and the access technology.*

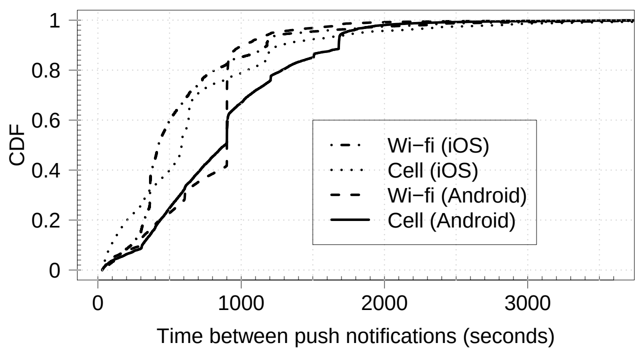


Figure 8: Distribution of the time between push notification messages in the wild. *The frequency of push notification messages is higher for the iOS devices in our dataset compared to the Android devices. Notification messages are less frequent over cellular networks compared to Wi-Fi networks.*

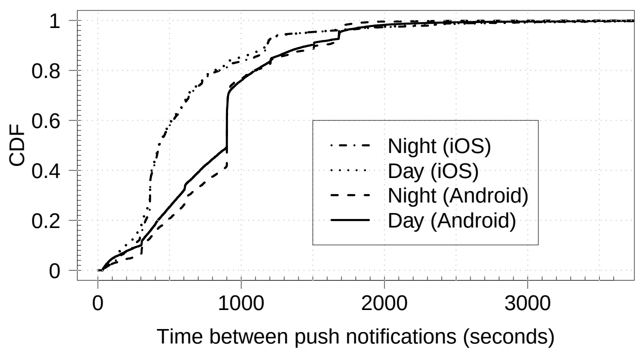


Figure 9: Impact of time-of-day on the push notifications. *The rate of notifications is agnostic of the time of the day for iOS and Android devices.*

not observe this difference when the Nexus S phone used the cellular data connection; we do not present the figure due to lack of space. [**TBD: AR: Why these difference – which applications stop coming up and so on**]. For the iOS devices, we observe an inter-arrival time at least 1700 seconds for that more than 75% of the notifications in Figure 7. We do not observe a significant difference between the inter-arrival times for the iPhone over cell and Wi-Fi and we do not present these results due to lack of space.

On analyzing the packets exchanged, we observe that 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. Similarly, 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.

In summary, we observe notifications consume very little data, less than 50 KB in 24 hours, on Android and iOS devices in their default state. The large time between successive notifications and the small amount of data exchanged implies they consume very little power. We also observe that iOS devices have a larger time between successive notifications compared to Android devices in the default state. Furthermore, the inter-arrival time between notifications for Android devices differs based on the device manufacturer and the access technology.

4.2 Notifications In The Wild

We now characterize our observations on the notifications we observed in the *mobWild* dataset. The objective of this analysis was to detail the frequency, traffic volume, and source of the notification services.

To analyze the frequency of notification messages, we plot the distribution of the time between successive push notification messages for Android and iOS devices over cellular and Wi-Fi networks in Figure 8. In Figure 8 we observe that a higher time between push notifications over cellular networks in comparison to Wi-Fi networks for iOS and Android devices. We also observe the Android devices in our dataset receive notifications less frequently compared to the iOS devices in our dataset. We also observe a heavy tail for the time between notification messages which implies potentially long idle intervals. [**TBD: Correlation between time between notification messages and size of the next idle time observed?**]

The notification messages could be in response to user actions, for example, a mail server might receive a notification of a new message. To analyze this impact, we notification messages received during two time intervals: from midnight to 6 am (night), and from 6 am to midnight (day). In Figure 9, we plot the distribution between successive notification messages for these two intervals. We observe that the Android and iOS devices appear 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

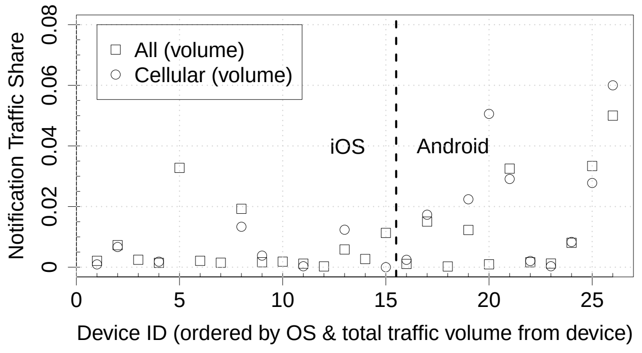


Figure 10: Traffic share of push notifications. *Push notifications are responsible for less than 5% of the traffic volume on most devices.*

periods. We observe notification messages were received by the device that had enabled this feature during the intervals their users had configured as *Do Not Disturb*.

To analyze the volume of notification messages, we plot the share of notification messages as a fraction of total traffic from the device in Figure 10. We observe that the notification messages are responsible for less than 4% of the traffic volume on most Android and iOS devices. In Figure 10, we also plot the share of notification messages when the device exchanged data over cellular networks. We observe that there is no significant difference in the traffic share of notification messages when the device used cellular traffic. [**TBD: Why do we care?**]

To further analyze the source of the notification messages we analyzed the DNS lookups that were performed before the connections for receiving notification messages was established. We observed that the servers that pushed content to iOS devices correspond to the DNS requests that match the pattern **courier.push.apple.com* and **courier-push-apple.com.akadns.net*. For the android devices we observe that the DNS requests match the pattern **talk.google.com*.

In summary, we detail the behavior of the notification services using controlled experiments and the *mobWild* dataset. We observe that push notifications consume a small fraction of the data and exhibit a heavy tail for the time between successive messages. We also observe that the frequency of notification messages was agnostic of the time of the day. Furthermore, notification messages were received by iOS devices even during the time interval the users had configured *Do Not Disturb*.

5. APPLICATION CHARACTERIZATION

To analyze applications, we need to first identify the applications that is responsible for the flows. Once identified we need to analyze the applications. For analysis we concentrate on 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. "With whom" is important to protect users from excessive tracking. 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.

5.1 Bandwidth and Radio Usage

5.2 Discussion

6. BEHAVIOR OF NETWORKS

7. RELATED WORK

Placeholder

8. CONCLUSION

Placeholder

9. REFERENCES

[1] Core Media Framework Reference. `https://developer.apple.com/library/mac/documentation/CoreMedia/Reference/CoreMediaFramework/`.

[2] GCM Architectural Overview. `http://developer.android.com/google/gcm/gcm.html`.

[3] Local and Push Notification Programming Guide. `http://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction.html`.

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

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

[6] 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.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol–http/1.1, 1999.

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

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

[10] 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.

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

[12] 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.