MobiPeek: Mobile Network Traffic Measurements using VPNs

Ashwin Rao
INRIA

Amy Tang
UC Berkeley

Justine Sherry
UC Berkeley

Adrian Sham
University of Washington

Sam Wilson
University of Washington

Shen Wang
University of Washington

Arnaud Legout
INRIA

Walid Dabbous
INRIA

Arvind Krishnamurthy
University of Washington

David Choffnes
University of Washington

ABSTRACT

Placeholder.

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.

Our contributions are as follows

- We present VPNs as a platform to provide a comprehensive perspective of mobile network traffic. Our VPN based platform has a low barrier to entry and can be used to perform measurement studies regardless of the mobile operating system, device type, wireless technologies, and service provider.
- We use this platform to compare the behavior of popular mobile applications over Android and iOS. We also compare the behavior of these application over Wifi and 3G.
- We provide a definition for a "Well Behaved App". We then check if the [**TBD: Number**] popular apps on Android and [**TBD: Number**] apps on iOS are *well behaved.*

2. MOTIVATION

The current mobile ecosystem is very opaque and it offers researchers a limited view into how mobile devices and the installed apps generate network traffic. This lack of transparency is currently being addressed by either instrumenting the mobile operating system (OS), instrumenting the binaries, static analysis of app binaries, or relying on ISP traces. In this section, we discuss each of these techniques and use their shortcomings to motivate the need *MobiPeek*, a VPN based platform for mobile measurements.

Instrumenting a mobile OS system provides researchers a fine grained view of the apps and OS in action. Taintdroid [4], one of the seminal works of instrumenting the Android OS, and its extensions have therefore been widely used to study Android apps in action. For example, Hornyack *et al.* [7] rely on Taintdroid to detail the internals of the 1100 most popular Android apps. The major shortcoming of instrumenting OSes is that the measurement results are limited to a specific OS version and apps written for that OS version. This implies that instrumenting OSes cannot be used for longitudinal studies that need to consider changes in not only the app code but also the OS code. Furthermore, because the app code is tightly coupled to the API provided by the underlying OS, the results obtained by instrumenting one mobile OS cannot be extrapolated to other mobile OSes. Instrumenting an OS also results in a high barrier to entry for studies that require participation of end users who may be unwilling to modify the underlying OS.

Instrumenting app binaries at predefined code points can be used to detail the behavior of a specific set of apps. One of the biggest advantages of instrumenting apps is its low barrier to entry because it does not require an OS modifications. This low barrier to entry was one the key motivations for the development of AppInsight [9]. Indeed AppInsight can provide developers improve the app quality, however, its scope is limited to the apps that have been instrumented, the marketplaces from where the apps are downloaded, and the OS version for which the instrumented app was written. Furthermore, each new version of the app needs to be instrumented.

Static analysis of the app code is used to study apps when the apps and the underlying OS are secured to avoid being tampered. For example, PiOS [3] was used to perform static analysis of 1400 IOS apps by static analysis. The authors of PiOS observe that the unique ID of the device is leaked by more than half of the apps they analyzed. A shortcoming of this study is that the PiOS can access the app binaries only after the iOS device is jail-broken, thus voiding the warranty of the device. Furthermore, like AppInsight [9], the results of PiOS are limited to the iOS operating system. [**TBD: Text for SPARTA project at UW.**]

ISP traces are useful to study mobile devices in the wild. Viallina-Rodriguez *et al.* [13] use an ISP trace of 3 million subscribers to detail the impact of ads and analytics on the mobile data and energy consumption. Similarly, Maier *et al.* [8] study the mobile traffic by looking at the DSL traces from a popular European ISP. However, these studies cannot provide a comprehensive view of the traffic from mobile devices because users can access the Internet using different ISPs depending on their location and the access technology used to connect to the Internet. For example, the home wi-fi and office wi-fi may be served from ISPs that are different from the ISP used for cellular data traffic.

[**TBD: In summary, portability, pervasiveness, ubiquitous, and deployable missing in existing platforms. In the next section we show that a VPN based platform can address these issues to get a network perspective of mobile devices**].

3. VPN BASED MOBILE MEASUREMENT PLATFORM

In this section, we enumerate the goals of a mobile measurement platform, show how VPNs can be used to achieve the described goals, and finally present empirical results that demonstrate the feasibility of *MobiPeek*, a VPN based platform for mobile network traffic measurement.

3.1 Goals

The primary goal for our platform is to provide comprehensive visibility into mobile networking traffic. To meet this goal, we further identify the following subgoals that address limitations of the existing platforms discussed in Section 2.

1. *Portable.* We want our solution to work regardless of operating system, access technology, and service provider.
2. *Pervasive.* For maximum transparency, our system should provide seamless visibility into all network traffic generated by devices. This means continuous monitoring over time and as users move with their
3. *Passive* For comprehensive and independent of user triggers we want our solution to passively perform measurements.
4. *Deployable.* Our solution should be easy to use, immediately deployable and incur reasonable costs (or none at all) for users. Furthermore, it should not incur the cost of warranty-voiding the device and by relying on techniques such as rooting a phone.

3.2 Description

We believe that VPNs can be used to setup a portable, pervasive, and deployable measurement platform for mobile device. Our motivation was the use of VPNs by executives "on the move" to securely connect to corporate servers with their mobile device. The use of VPNs by corporate clients gave us hints towards considering VPNs as portable and deployable platform. Further investigation showed us that Mobile OSes expose features that can make them pervasive.

3.2.1 Mobile Devices

All iOS devices (version 3.0 and above) come with 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. We use this feature to enforce our iOS devices to use a VPN tunnel to connect to the Internet.

Android version 4.0 and above comes with native VPN support. Unlike iOS, Android does not offer an equivalent of *VPN On-Demand*; however, Android provides an API that allows an user space app to manage VPN connections. We modify the open source StrongSwan VPN client [10] to ensure that the VPN reconnects each time the preferred network changes (*e.g.*, when a device switches from cellular to Wi-Fi). As of Android 4.2, Android supports "Always On" VPN connections that uses VPNs to tunnel all the data traffic. [**TBD: Text on Issues with Always ON**].

3.2.2 VPN Server

Strongswan, Openswan, and OpenVPN are some of the popular VPN daemons that can be used to manage VPN tunnels. We use Strongswan because it is open source and, to the best of our knowledge, it is the only open source solution that can use the IPsec services of the Linux kernel without any kernel modifications. IPsec is important because the *VPN On-Demand* feature of iOS is supported only for VPN tunnels that use IPsec. Strongswan is supported on vanilla Linux operating systems which ensures that VPN tunnels can be managed using *off-the-shelf* software and hardware.

In summary, the active use of VPNs to access corporate networks regardless of the ISP, access technology motivated us to consider VPNs. Mobile devices can connect to open source VPN implementation that can run on off-the-shelf servers. Our platform relies on VPNs to tunnel and capture the mobile data traffic. We use the open source Strongswan daemon to manage VPN tunnels. We use tcpdump to capture the packets on the server that runs Strongswan. Our servers have been deployed at the University of Washington and Inria and the mobile devices belong to users participating in an IRB-approved study.

[**TBD: How we capture and why this is important to label flows to access technology and ISP?New instance of tcp dump for each flow. Each time a tunnel is created separate instance of tcpdump used – required to isolate flows. We log the IP address from which tunnel is created. We use this IP to get the details of the ISP. We then look up the ISP and label the connection to be wifi or celluar. Pros and cons of this approach.**]

3.3 Feasibility

We now show that a VPN based platform is portable, pervasive, deployable, and incurs a small overhead in terms of latency, power, and bandwidth.

3.3.1 Portable, Pervasive, Passive, and Deployable

Android, BlackBerry, Bada, and iOS all support VPNs natively, representing more than 86% of the mobile device market[6]. These devices support VPN tunnels over Wi-Fi and the cellular interface. Furthermore, to satisfy their corporate clients, very few ISPs are known to block VPN traffic to flow through their networks.

We use the "VPN On-Demand" feature provided by iOS to ensure that our system is pervasive. Android provides an API that allows user space apps to manage VPN connections. We modify the open source StrongSwan VPN client [10] to ensure that the VPN reconnects each time the preferred network changes (*e.g.*, when a device switches from cellular to Wi-Fi). As of Android 4.2, Android supports "Always On" VPN connections that uses VPNs to tunnel all the data traffic. [**TBD: Text on Issues with Always ON**].

Manually configuring a VPN generally requires filling out five fields on an Android phone, and the VPN configuration can be distributed using a single file on iOS. These configurations are primarily required to drive the key exchange algorithms that establish the VPN tunnels. This simplicity is important because it can facilitate realistic measurement studies with end users.

In summary, native VPN support along with a set of features exposed to by mobile OSes to manage VPNs offer a solid foundation to build a portable, pervasive, passive, and deployable platform for mobile measurements.

3.3.2 Latency Overheads

Number of round trip times to establish the connection. [**TBD: Text from results from Adrian and Sam**]

3.3.3 Power Overheads

[**TBD: TODO if needed**]

3.3.4 Bandwidth Overheads

Number of packets to establish the connection

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

For our measurements, we capture the encrypted packets exchanged by our servers and the clients that use *MobiPeek*. We performed the packet capture for [**TBD: Number**] days during which [**TBD: number**] devices tunneled their traffic via our servers. During this time interval we also capture the packets that were encapsulated in the IPsec packets. We use these samples to compute the increase in the amount of bytes transferred due to encapsulation and the keep-alive messages. During the 14 day period we observe that the median of the increase to be [**TBD: number**]%, with a maximum increase of [**TBD: number**]%

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

4. DATASET DESCRIPTION

In this section, we describe the three datasets: *mobAll*, *mobCompare*, and *mobileExpt*. We use these datasets in our studies that we present in the subsequent sections.

[**TBD: Run genDataSetDescription.R to get these numbers**] The *mobAll* dataset consists of mobile data traffic traces from **!23!** devices that belong to **!18!** users who are volunteers of an IRB approved study. This dataset consists of **!8!** iPhones, **!3!** iPads, **!1!** iPodTouch, **!10!** of Android phones, and **!1!** of Android tablet. We consider an Android device that accesses the Internet by using only Wi-Fi to be a tablet. The Android devices used to generate this dataset include the Nexus, Sony, Samsung, and Gsmart brands. The users of these devices are spread across France and USA. We observed connections from **!52!** different ISPs. This dataset consists of **!171!** days of data that was captured on our servers and for each device the number days varies from **!5!** to **!171!** with a median of **!34!** days.

The *mobCompare* dataset consists of data from two users who had three and two devices respectively. The three devices that belong to one of the two users consists of a two smart phones and a tablet, while one smartphone and a tablet belong to the second user. We use this dataset to compare the behavior of popular apps and to detail the behavior of devices when the device is kept idle. The data set consists of **!number!** of days of data from each device and [**TBD: number**] of days for which data traffic was seen for all the 24 hours.

The *mobileExpt* dataset contains the traffic traces from an Android device and an iOS device that were used to perform a controlled experiment on popular application. We tested **!number!** of Android applications and **!number!** of iOS application for this study. [**TBD: How we decided this list**]. [**TBD: How we performed the test**] [**TBD: other attributes of this dataset**].

5. MEASUREMENT RESULTS

[**TBD: In this section we ...**]

5.1 Descriptive Statistics

We now use several descriptive statistics to summarize our *mobAll* dataset. We use these descriptions to support our claim that *MobiPeek* can be used as a portable, pervasive, and deployable platform for passive measurements of mobile network traffic.

We use bro to parse the pcap files. Bro relies on transport layer protocols and port numbers to classify the traffic. We build on this initial classification provided by bro to broadly classify IP flows based on the protocols as TCP, UDP, or other. Flows such as ICMP that are neither TCP nor UDP have their protocol classified as other other. Such flows have their *proto* field as other in Table 1 and Table 2. Bro further classifies flows that TCP port 80 and port 8080 as HTTP flows. SSL flows include those that use port 443 (HTTPS), 993 (IMAP), and other ports that are known to use SSL. All TCP flows that are neither HTTP nor SSL are classified as other. Such flows have their *service* fi eld as other Table 1 and Table 2. All UDP flows regardless of the applications using UDP such as DNS and Skype, are classified as UDP.

5.1.1 *Comparison of Devices*

One advantage of using *MobiPeek* is that we can detail and compare the behavior of mobile devices regardless of the underlying operating system. We now use Table 1 to directly compare Android and iOS, focusing on the key protocols used by the devices.

Our first key observation is that HTTP is responsible for the majority TCP traffic volume in bytes, **!63.13%!** for Android and **!80.28%!** for iOS. This observation is inline with previous results that report HTTP to be the dominant protocol used by mobile devices [5, 8]. We also observe that the majority of TCP flows are sent over SSL, **!43.77%!** of total flows for Android and **!32.18%!** of total flows for iOS . We believe this occurs across platforms because the majority of flows come from e-mail, messaging and social networking, all of which use secure channels regardless of the OS. We discuss some of these popular application in [**TBD: section**].

For the Android devices in the *mobAll* dataset, **!96.45%!** of the UDP flows that account for **!70.92%!** of the UDP bytes are due to DNS request. The rest of the traffic is due to other applications such as Skype. Similarly,

Protocol	Service	Android		iOS	
		Flows	Bytes	Flows	Bytes
TCP	HTTP	13.55	63.13	14.64	80.28
TCP	SSL	43.77	31.36	32.18	18.98
UDP	-	37.23	1.09	47.32	0.55
TCP	other	2.24	4.30	1.55	0.15
Other	-	3.19	0.10	4.03	0.01
total		100.00	100.00	100.00	100.00

Table 1: Percentage of flows and bytes from iOS and Android devices. [**TBD: Verify total 100 for final results**] *SSL is responsible for the majority of TCP flows from iOS and Android devices.*

!80.98%! of UDP flows that account for **!66.29%!** of UDP bytes from iOS devices are due to DNS requests.

We observe 0.1% of the traffic volume was classified as *other* for Android device which is larger than the 0.01% observed for iOS. This is because the Android users in our dataset tend troubleshoot network connectivity issues using applications that perform pings and traceroutes.

On balance the number of bytes transferred per unit time in Android, and the number of flows, is larger than the same for iOS. We found that iOS devices generated about **!number!** of traffic per hour while Android devices generated about[**TBD: number**] MB/hr, and increase of **!number%!**. Further, Android devices contributed more flows (**!number!** per hour vs. **!number!** per hour), an increase of **!40%!**. It is difficult to account for the impact of user behavior on device-generated traffic; that aside, one explanation is that Android devices generate more bytes and flows due to the relatively permissive API for running code in the background on Android. In contrast, iOS quickly kills processes that are in the background, preventing them from generating network traffic after only a few seconds of losing the foreground. We discuss this [**TBD: section**]

[**TBD: In summary,**]

5.1.2 *Comparison of Access Technology*

MobiPeek allows us to passively monitor traffic regardless of the access technology used by the mobile device. We use this feature of *MobiPeek* to provide a direct comparison of how the different access technologies are used by our users.

We use the following technique to obtain the access technology used by the mobile devices when using the VPN services. To map the IP addresses of the mobile devices to the AS and ISPs used by the mobile devices while establishing the VPN tunnel, we rely on the *WHOIS* databases of *cymru.com* and *utrace.de*. We use the *WHOIS* responses that includes the AS and ISP, and inputs from the users, to manually classify the connection used as either Cellular and Wi-Fi. This clas-

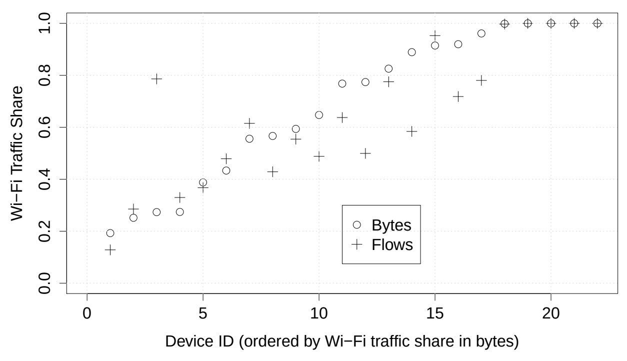

Figure 1: Traffic share of Wi-Fi as a fraction of total traffic from a device.

sification technique fails when the same AS is used for Cellular and Wi-Fi traffic. We would like to point out that we were not able to observe such behavior from the ASes used by our users. However, this technique shall wrongly classify the access technology when the Wi-Fi access point internally uses cellular services to connect to the Internet. In such cases, despite the use of Wi-Fi by the mobile device we classify the flows to be cellular flows because our server sees the traffic to be originating from an AS that serves cellular connections.

In Figure 1 we present the share of Wi-Fi as a fraction of the total traffic generated by the user. The devices are sorted according to the share of Wi-Fi traffic as a fraction of the total traffic from the device. The devices with device IDs from !18! to !23! are devices that do not have a cellular data plan associated with them device: !3! iPads, !1! iPodTouch, and !1! Android tablet. The diversity in Wi-Fi traffic share in terms of flows and bytes indicates the diversity of mobile device usage. For example, device ID !3! is used by a user who prefers to use cellular network to listen to music, watch videos, and troubleshoot the cellular network connectivity issues using tools such as speedtest; the primary use of the device is used to read emails and access social networks when on Wi-Fi. Media content such as music and videos have a higher share of traffic volume per flow, therefore the share of cellular traffic in higher for device ID !3!. Similarly, device ID !1! is used by a user who prefers to use cellular networks.

In Table 2 we further classify the protocols and services that are responsible for Wi-Fi and cellular data traffic with the help of the techniques used to generate Table 1. The first key observation is that the portion of HTTP bytes sent over Wi-Fi and cellular are significantly different. Upon further inspection, we found that Wi-Fi is the preferred medium to transfer media content, which generates relatively large flows. For example, apps such as Google Plus image backup on Android allow users to upload their images only over Wi-Fi. In

Protocol	**Service**	**Wi-Fi**		**Cellular**	
		Flows	**Bytes**	**Flows**	**Bytes**
TCP	HTTP	16.14	81.59	10.99	57.08
TCP	SSL	32.56	16.97	43.99	39.59
UDP	-	46.29	0.52	38.56	1.25
TCP	other	1.43	0.89	2.46	1.97
Other	-	3.55	0.02	3.96	0.09
total		100.00	100.00	100.00	100.00

Table 2: Percentage of flows and bytes using Wi-Fi and Cellular as access technology. [**TBD: Verify total 100 for final results**] *The share of SSL traffic over Cellular networks is larger than its share over Wi-Fi in terms of bytes and flows*

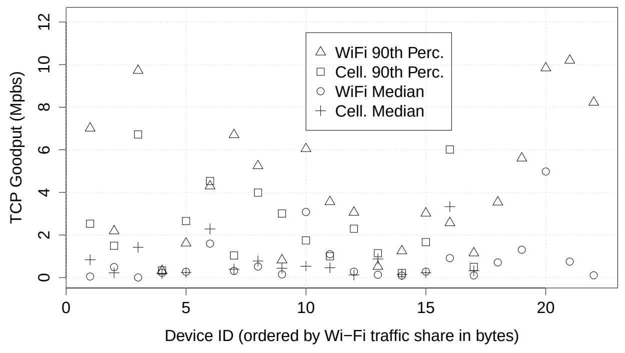

Figure 2: Goodput of TCP flows over Wi-Fi and cellular networks.

line with the difference in HTTP traffic ratios, we note that SSL flows are dominant type in cellular connections. We also observe a smaller number of UDP flows for cellular networks. [**TBD: Why do we observe less??**].

For each device in the *mobAll* dataset, we present the goodput of TCP flows that exchanged more than 512 kB of data in Figure 2. [**TBD: cite [1] if needed for 256 kB**]. We use the goodput because it a good estimate of how the applications perceive the quality of the network. The devices are sorted according to the same technique used to sort the devices in 1. For device 16 we observe higher data rates for cellular compared to Wi-Fi because the user sparingly uses the cellular connection (see 1). For device 6 we observe similar values for the TCP goodput that the median and the 90th percentile. This highlights opportunities for proposals such as 3G onloading [12].

All the users except used at most 1 cellular ISP apart from two users who used 2 and 3 respectively. However, for Wi-Fi the median number of ISPs observed was 4 with a maximum of 24 for one user. The user who contributed 24 distinct ISPs used *MobiPeek* when travelling across 6 different countries. This implies *MobiPeek* was able to capture traffic of users "on the move." Campus wide studies such as [**TBD:**], studies on DSL networks [8], and studies limited to traces from one specific ISP [13] are not able to capture this behavior.

[**TBD: In summary,**]

5.2 Case Studies

5.3 Controlled Experiments

5.4 Discussion

6. RELATED WORK

Placeholder for the papers. [11]
[2]
[13]
[5]

7. CONCLUSION

Placeholder

8. REFERENCES

[1] HTTP Archive. `http://httparchive.org`.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones. In *Proc. of IMC*, page 280, New York, New York, USA, Nov. 2009. ACM Press.

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

[4] 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.

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

[6] Gartner smart phone marketshare 2012 Q1. `www.gartner.com/it/page.jsp?id=2017015`.

[7] 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.

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

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

[10] strongswan VPN client. `http://play.google.com/store/apps/details?id=org.strongswan.android`.

[11] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who Killed My Battery: Analyzing Mobile Browser Energy Consumption. In *Proc. of WWW*, pages 41–50, 2012.

[12] N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, and K. Papagiannaki. When david helps goliath: The case for 3g onloading. In *Proc. of HotNets*, 2012.

[13] 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.