MobiScope: Practical Mobile Internet Traffic Monitoring

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

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

1. INTRODUCTION

Our contributions are as follows

- We posit that VPNs can be used build a practical platform to monitor mobile Internet traffic regardless of the mobile operating system, device type, access technologies, and service provider.
- We present *MobiScope*, a practical platform for comprehensive monitoring of mobile Internet traffic. *MobiScope* builds on existing functionality provided by Mobile OSes to manage VPN tunnels.
- We use *MobiScope* to compare the behavior of popular mobile applications over Android and iOS. We observe [**TBD: values come here**]. We also compare the behavior of these application over Wifi and 3G.
- [**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.**]

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

2. MOTIVATION

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

Instrumenting a mobile OS system using tools such as Taintdroid [2] and AppFence [4] provides researchers a fine grained view of the apps and OS in action. However, this fine grained view comes at a high cost of jailbreaking and warranty voiding of the devices. Most end users shall not be willing to pay this cost. Instrumenting an mobile OS is therefore not practical for measurement studies and experiments that require end user participation. Furthermore, longitudinal studies that detail the impact of OS code changes and application code changes cannot be performed by instrumenting OSes.

Instrumenting app binaries using tools such as AppInsight [7] can be used to characterize the network traffic from mobile applications. Indeed, AppInsight provides a detail analysis of mobile applications. However, in terms of the network footprint of the app, the scope of AppInsight is limited to the instrumented apps, the marketplaces from where the apps are downloaded, and the OS version for which the app was instrumented. Furthermore, each new version of the app needs to be instrumented to characterize the impact of the changes.

Static analysis of the app code can be used to study mobile application whose code cannot be instrumented. For example, PiOS [1] was used to perform static analysis of 1400 IOS apps. A shortcoming of PiOS is that access to the app binary is possible only if the device is jail-broken, thus voiding the warranty of the device. Furthermore, like AppInsight [7], the results of PiOS are limited to the iOS operating system. [**TBD: Dave Text for SPARTA project at UW.**]

ISP traces are useful to study mobile devices in the wild. Vallina-Rodriguez *et al.* [11] use an ISP trace of 3 million subscribers to detail the impact of ads and analytics on mobile Internet traffic and energy consumption. Similarly, Maier *et al.* [6] study the mobile traffic by looking at the DSL traces from a popular European ISP. However, the data used in these studies is limited to the ISP that provided these traces. Mobile devices can use different ISPs depending on their location and the access technology used to connect to the Internet. For example, the home Wi-Fi and office Wi-Fi may be served from ISPs that are different from the ISP used for cellular data traffic. Therefore, traces from a single ISP are expected to have a limited view on the traffic from the mobile devices.

ISPs can interfere with the Internet traffic to inject javascript code for advertisements and analytics. This problem of ISP interference was highlighted by Reis *et al.* [8]. The authors demonstrated that inflight changes made by ISPs tend to introduce vulnerabilities such as overflows and cross-site scripting (XSS) attacks. They proposed and deployed *Web Tripwires*, a Javascript code that detects in-flight page changes. The main limitation of *Web Tripwires* is that Web sites are required to modify their content to include a tripwire that can detect ISP interference. *Web Tripwires* are therefore not practical because they require support from the Web site maintainers.

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

3. TRAFFIC REDIRECTION TO MONITOR MOBILE INTERNET TRAFFIC

[**TBD: Should we explicitly mention traffic redirection as a tool?**] In this section, we first detail our goals. We then show that traffic redirection can be used to achieve our described goals in a feasible manner.

3.1 Goals

Our primary goal is *to monitor all the Internet traffic from mobile devices regardless of the operating system, wireless access technology, and the ISPs used by the mobile device.* We now present the following sub-goals we believe are essential to meet this goal.

1. *Portable.* We want to maximize the number of OSes and brands that are included in measurement studies.
2. *Pervasive.* We want our measurements to be agnostic of the location, access technology, and ISPs used by mobile devices.
3. *Passive.* We want to monitor traffic even when the devices are *idle.*
4. *Deployable.* We want a low barrier to entry to ensure that end users can participate in our studies.

In summary, we define our goal using portability, pervasiveness, passiveness, and deployablity as its building blocks. [**TBD: These points need to be revisited to incorporate Trip wires.**]

3.2 Description

We now show that traffic redirection is the best way to reach our goals. We use two forms of redirection: a VPN based redirection to detail the network traffic characteristics of mobile device, and a transparent proxy based redirection to detail ISP interference of mobile Internet traffic.

3.2.1 VPN based Traffic Redirection

We were motivated to consider VPNs because corporate executives are known to connect to securely to their corporate servers using VPNs regardless of their location and available access technology. The fact that corporate clients used VPNs gave us hints towards its portability and deployability. We now show that VPNs can be pervasive and ideal to monitor mobile Internet traffic from end users.

Android, BlackBerry, Bada, and iOS all support VPNs tunnels over Wi-Fi and the cellular interface. 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. Using trial-and-error, we discovered that VPN On-Demand uses suffix matching to determine which domains require a VPN connection. We use this knowledge to configure *VPN On-Demand* such that iOS devices use a VPN tunnel to access the Internet. 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 con-

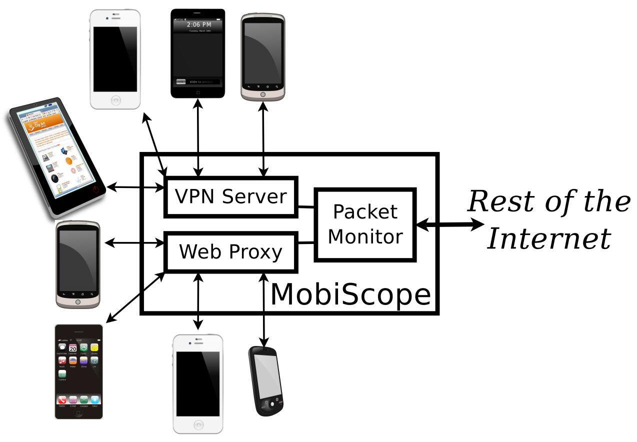

Figure 1: *MobiScope* uses traffic redirection to monitor mobile Internet traffic. *MobiScope requires mobile clients to redirect their traffic through a server that monitors Internet traffic. VPN based redirection is used to characterize mobile traffic sans ISP interference. To detect ISP interference*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

nections that uses VPNs to tunnel all the data traffic. [**TBD: Dave:text on Always ON from inputs from Adrian**].

MobiScope uses Strongswan [9] because it is the only open source solution that can use the IPsec services of the Linux kernel *without any kernel modifications*. We emphasize on IPsec because the *VPN On-Demand* feature of iOS is supported only for VPN tunnels that use IPsec and the IKEv1 [3] key exchange protocol. Furthermore, Strongswan also supports the faster IKEv2 [5] key exchange protocol that is supported by Android clients.

3.2.2 Redirection using a Transparent Proxy

[**TBD: Dave add text here**]

3.2.3 Traffic Monitoring Setup

As shown in Figure 8, mobile devices redirect their Internet traffic through a *MobiScope* server that monitors all their Internet traffic. VPNs are used to monitor all Internet traffic from mobile devices while a transparent Web proxy to detail interference by mobile ISPs.

Deploying VPNs on mobile clients is simple for end users primarily VPNs are natively supported by popular mobile OSes. The Android users need to install a certificate and fill our five fields while iOS users need to install a configuration file. Once the configurations are stored, all Internet traffic from the mobile device flows through *MobiScope*. *MobiScope* uses Strongswan to manage VPN tunnels of *MobiScope*. This simplicity is important for practical and realistic measurement studies with end users. We use tcpdump to monitor the

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

Figure 2: Overview of the Web tripnet

traffic that passes through *MobiScope* servers.

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

In summary, mobile VPNs are portable and deployable because they are natively supported by popular mobile operating systems. We build on existing features provided by iOS and Android to make sure that the VPN tunnels are pervasive and are created passively. We rely on the open source Strongswan VPN daemon to manage VPN tunnels which makes *MobiScope* deployable. We plan to release the source code to configure mobile Internet traffic using *MobiScope*.

3.3 Feasibility

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

3.3.1 VPN Latency Overheads

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

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

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

3.3.2 *Data Consumption Increase when using VPN*

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

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

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

3.3.3 *Power Overheads when using VPNs*

[**TBD: Daves results**]

[**TBD: In summary, we show *MobiScope* is feasible to build and deploy**]

3.3.4 *Impact of using Transparent Proxy*

[**TBD: Dave: Text comes here**]

3.4 Discussion

We now discuss some shortcoming of using traffic redirection

1. Traffic redirection implies that Web sites that use their clients IP addresses to offer custom services shall respond according the IP address of the *MobiScope* server. This can affect the quality of the results we observe in our datasets.
2. Some ISPs are known to block VPNs. During our measurement we observed one such ISP that blocked VPN tunnel creation requests from one of our clients.
3. We observe that mobile devices currently create a VPN on at most one wireless interface. This implies that *MobiScope* shall not be able to monitor traffic when the mobile device simultaneously use more than one wireless interface.
4. We performed controlled experiments to test IPv6 support. We observe that though iOS and Android support IPv6 they currently cannot tunnel IPv6 traffic through VPN tunnels.
5. When using VPNs, *MobiScope* shall not be able to monitor the traffic between the mobile device and access point used to connect to the Internet.

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

4. METHODOLOGY AND DATASET

In this section, we detail the data collection methodology of *MobiScope* and describe the datasets we collected. Our objective for monitoring mobile Internet traffic was to understand the Internet usage of mobile devices and the ISP interference of mobile Internet traffic. To achieve this objective we performed controlled experiments and also allowed real users to redirect their traffic through our VPN servers.

4.1 Controlled Experiments

We performed controlled experiments to detail mobile Internet traffic is affected by mobile operating systems, background processes running on mobile devices, applications installed on the mobile devices, and interference of mobile ISPs.

The applications running on iOS devices receive notifications from Web services using *iOS push notification*. The iOS push notifications have received little research attention due to the closed-source nature of iOS. We ran controlled experiments using an iPhone to detail the behavior of *push notificaton* and to the best of our knowledge we are the first to characterize iOS push notification.

[**TBD: Ashwin: Paragraph on Android comes here**]. This text is a placeholder text for this paragraph. Please replace this text.

[**TBD: Justine: I need your help to rewrite the text here. I have put some crappy text as placeholder.**] Mobile applications have received attention for leaking personally identifiable information (PII) to ads and analytics sites [4, 1][**TBD: papers**] We performed controlled experiments using an Android device that contained dummy user credentials for popular social networking sites and dummy contact information. We then used [**TBD: Monkey**] to automatically test [**TBD: number**] applications. We selected these applications because [**TBD:**].

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

[**TBD: Dave: Text Here**]

4.2 In The Wild

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

The 25 devices that contribute to the *mobAll* dataset consists of 9 iPhones, 4 iPads, 1 iPodTouch, 10 Android phones, and 1 of Android tablet. Though *tablets* can access the Internet via a cellular data connections, for the *mobAll* we consider tablets to be devices that only use Wi-Fi to access the Internet. The Android devices in this dataset include the Nexus, Sony, Samsung, and Gsmart brands. This dataset consists of 190 days of data that flowed through our VPN servers; the number days for each user varies from 5 to 176 with a median of 33 days.

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

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

Based on the above classification of access technology and ISPs, our dataset consists of data traffic from 54 distinct ISPs, of which 10 provided cellular services. Of the 19 devices that used cellular data, we observed that 16 devices restricted their cellular data traffic to one ISP each; the other three users used the services of four, two, and two ISPs respectively. We observed that the devices in our dataset used a higher number of Wi-Fi ISPs. We observed a median of 4 Wi-Fi ISPs per device with a maximum of 24 Wi-Fi ISPs that were used by one device. This observation confirms our intuition that studies based traces from a single ISP [6, 11], shall not be able to analyze how specific users use mobile devices.

4.3 Discussion

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

5. NETWORK CHARACTERISTICS OF OPERATING SYSTEM SERVICES

Mobile operating systems provide APIs and OS level services to optimize network usage. For example, iOS applications can use the Apple Push Notification Service (APNS) to receive notifications from the Internet. Similarly, iOS and Android APIs limit the background activity of applications to limit the network usage and extend the battery life of mobile devices. In this section we perform a set of controlled experiments to detail the network characteristics of the OS services. The questions that we answer in this section are as follows.

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

5.1 Network Characteristics of [TBD: Shipped/Factor Reset] Devices

We now detail the network characteristics of devices that contain only the pre-installed applications. We use the following questions as to guide our analysis

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

We began this experiment by first performing a factory reset on an iPod Touch, an iPad, an Samsung Galaxy SIII, and a Google Nexus S Phone. All the devices had their batteries fully charged before the factory reset was performed. After the factory reset, we allowed these devices to connect to the Internet using our Wi-Fi hotspot. We use the same dummy email account on each device as the primary account for that device. We ran tcpdump on our hotspot to monitors the Internet traffic from these devices for 48 hours. For the first 24 hours each of the devices had a dummy gmail account as the primary device account. For the final 24 hours we added a facebook account to the list of accounts. We use the data collected as a rough estimate on the minimum data traffic that is generated by the devices.

In Figure 3 we observe that the devices exchange a significant amount of data during the first few minutes of the first boot. Despite the same account we observe

[**TBD: x axis is the hour, y axis the amount of data downloaded in that hour in a log scale. log scale to account for the first hour.**]

Figure 3: Time evolution of traffic volume. *We observe that in the first few minutes of connection establishment, the devices download the maximum amount of data.*

Device Type	**Total Volume**	**TCP Volume**	**TCP Flows**	**Median Flows/Hour**
iPod Touch	a	a	a	a
iPad	a	a	a	a
iPhone	a	a	a	a
Samsung Galaxy SIII	a	a	a	a
Google Nexus S	a	a	a	a

Table 1: Traffic summary of devices after factory reset.

that [**TBD: the iPad**] has the significantly larger traffic volume compared to the other devices. We also observe that after the first hour, the traffic volume is minimal. We observed that UDP corresponded to a maximum of !**number**$_{verify}$! of flows, and the !**x%**$_{verify}$! UDP flows were DNS requests observed before TCP connections were established.

As show in 1 we observe that the median number of flows per hour is negligible. For the iOS devices we observe that the primary contributor for the traffic is the iOS push notification. We detail the characteristics in the rest of this section. During the 48 hour interval we observed that the !**device**$_{verify}$! produced the largest number of TCP and UDP flows !**number**$_{verify}$! of flows.

[**TBD: Specific results come here. Motivate iOS push here for iOS devices.**] Placeholder for para on specific results.

[**TBD: Discussion comes here**] Placeholder for discussion.

5.2 iOS Push Notifications

We now present the results of a detailed case study on the Apple Push Notification Service (APNS) that is used to push notification to iOS devices. We focus on APNS because it represents an OS-managed service on an operating system that received little research attention due to its closed-source nature. We use the following questions as to guide our analysis:

1. What is the network impact of push notifications and does the measured activity coincide with published documentation?
2. Are we able to capture the relevant data using traffic redirection?
3. What is the impact of traffic redirection using VPNs on the iOS push notifications?
4. Can we use network traffic alone to infer the state of the device?

The Apple Push Notification Service (APNS) implements push for iOS. The documentation for APNS provides limited details about the implementation, but does specify expected behavior (*e.g.*, push connections are established over cellular connections even if Wi-Fi is available)

We explicitly verified all provided documentation and confirmed that all statements are true with the exception of the notification behavior with an iPad. The documentation states that the iPad will always remain associated with a Wi-Fi AP, even if it is not plugged in. Our experience shows this is not the case on an iPad 2.

[**TBD: Ashwin: Text from Arnauds experiments**]

[**TBD: Ashwin: x axis is the hour of the day, y axis is the median number of push notifications received. plot for 4 users, 2 iphones and two 2 ipads**]

Figure 4: Push notification in the wild (time of day).

[**TBD: Ashwin: x axis is the device id sorted by frequency of push notifications, y axis is the median number of push notifications received per hour. Data for 14 users that includes iOs device and iPhone, two points for each user: one for cellular and one for wifi**]

Figure 5: Frequency of push notification in the wild.

5.3 iOS Push in the Wild

We now use the data from our *mobAll* dataset to further investigate the behavior of iOS push over time for !**four characteristic users**$_{verify}$!. The objective of this analysis was to answer the following questions

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

5.4 Discussion

6. CHARACTERIZE APPLICATIONS

1. How does an application behavior change with operating system and access technology?

[**TBD: Ashwin: x axis is the hour of the day, y axis is the number of push notifications received. plot for 4 users, 2 iphones and two 2 ipads**]

Figure 6: Push notification in the wild (time of day).

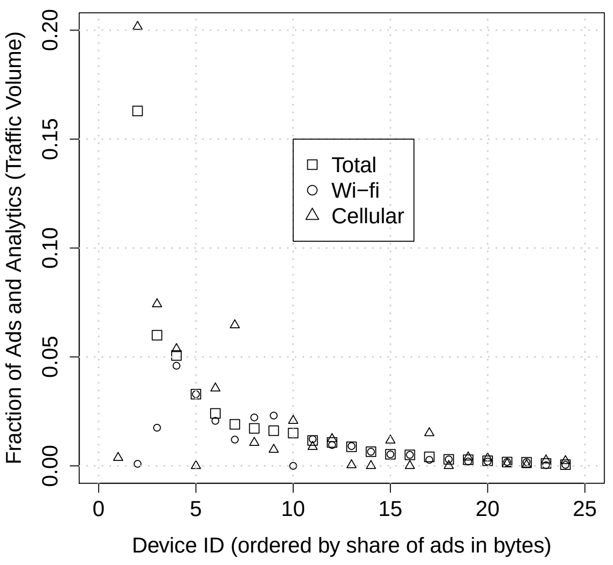

Figure 7: Fraction of traffic volume because of Ads and Analytics. *[***TBD: Check for id1 and id25***]*

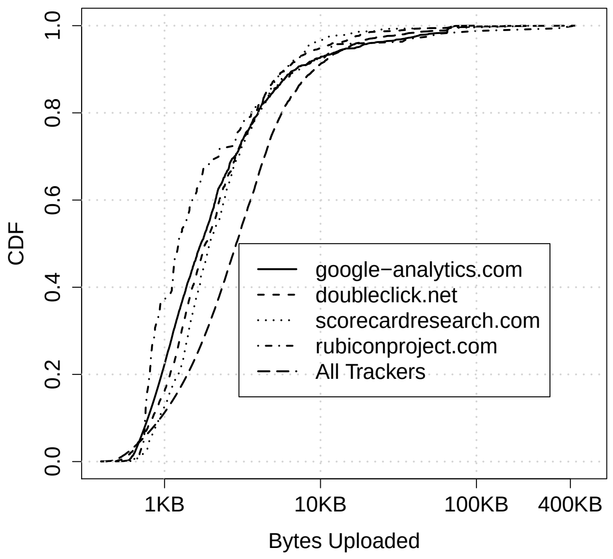

Figure 8: Distribution of bytes uploaded by ads and analytics sites. *The distribution of bytes uploaded by all ads and analytics sites and the top four ads sites based on traffic volume across all users.*

2. How intrusive are popular applications?

6.1 Characterize Facebook Applications

Why Facebook was chosen?
What do we observe ?
What do we see in the User Agent Fields.

6.2 Characterize Ads and Analytic Traffic

How do we characterize ads and Analytics.
What fraction of traffic is due to ads and analytics.
Breakup over Cell and Wifi and Android and iOS.
What more do we observer.
Now motivate for the next section saying we observe PII leaks.

6.3 Identify Instrusive Applications

Why did we do this test?
How did we perform our experiments?
What did we observe?
What is different from previous observations?
What insights do we gain from our results?

7. BEHAVIOR OF NETWORKS

7.0.1 Controlled Experiments

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

Table 2: The top 10 ads and analytics sites that tracked the devices in our dataset. *Two trackers,* doubleclick.net *and*google-analytics.com, *were tracking all the 25 devices in our dataset.*

7.0.2 *In the Wild*

8. RELATED WORK

Placeholder

9. CONCLUSION

Placeholder

10. REFERENCES

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

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

[3] P. Hoffman. Rfc 4109: Algorithms for internet key exchange version 1 (ikev1). *IETF Request For Comments, http://www.ietf.org/rfc/rfc5996.txt*, 2005.

[4] 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.

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

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

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

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

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

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

[11] 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.