Meddle: Transparency and Control for Mobile Network Traffic

Ashwin Rao
Inria

Amy Tang
UC Berkeley

Justine Sherry
UC Berkeley

Arnaud Legout
Inria

Walid Dabbous
Inria

Arvind Krishnamurthy
University of Washington

David Choffnes
University of Washington

ABSTRACT

Today's mobile systems make it difficult to obtain visibility into network traffic and provide few mechanisms for changing how devices and apps use the Internet. As a result, we have a poor understanding of – and limited ability to improve – network performance, battery consumption, data-quota usage, and privacy in the mobile environment. In this paper, we address these issues using *Meddle*, a platform that improves transparency for mobile networking traffic and enables fine-grained control over flows generated by mobile devices.

Meddle achieves this by using virtual private networks (supported by nearly all devices) to forward traffic from devices to a server that uses software-middlebox techniques to modify, shape and/or block network traffic before forwarding it to the destination. Unlike previous work, *Meddle* provides the ability to characterize and control mobile-device traffic on user devices regardless of the OS, carrier or access technology – without rooting the device or modifying the default OS. Further, *Meddle* allows us to experiment with new techniques for interposing on mobile traffic in a way that improves performance, resource consumption and privacy.

This paper describes the architecture of *Meddle* and results from its initial deployment, showing that it imposes a modest overhead while providing users with services previously unavailable on mobile devices. We further use the network flows traversing *Meddle* to reveal unexpected and alarming network behavior of mobile apps made visible through our approach. Last, we discuss how *Meddle* can address these and other issues in the mobile environment without requiring app or OS modification.

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. Despite a large collection of privacy, policy and performance issues in mobile networks [7, 12, 24, 16], users and researchers are faced with few options to characterize and address them.

In this paper, we demonstrate that we can provide the necessary framework to simultaneously address many of these issues for users and researchers by using middleboxes accessible through virtual private network (VPN) tunnels, an approach we call *Meddle*. *Meddle* works for nearly all mobile devices out of the box: Android, iOS, Windows 8 (non-ARM) and Blackberry all support VPNs, thus providing a portable mechanism to tunnel traffic to servers outside of the carriers' control regardless of the mobile device's network. Once packets arrive at VPN servers, we use a variety of software-middlebox [22] approaches to transform traffic to and from mobile devices. Together, these features enable the following new opportunities for characterizing and controlling network traffic in the mobile environment.

First, *Meddle* provides a portable, pervasive, passively-gathered view of mobile network activity from mobile devices. It is portable in that VPNs work on nearly all devices and they capture traffic regardless of which carrier or access technology (*e.g.*, cellular or Wi-Fi) is being used. It is pervasive in that *Meddle* ensures a device's VPN is always connected when devices access the Internet, offering a comprehensive and continuous view of network traffic generated by devices. By tunneling traffic to a server we control, *Meddle* enables low-cost passive monitoring of traffic naturally generated by users, OSes and apps on their own devices.

Second, *Meddle* provides a new point of control over mobile network traffic. This enables researchers to investigate what-if scenarios for the impact of new middleboxes as if they were deployed in carrier networks. These include new app-accelerators, mobile-specific security filters and protocol manipulation to improve power consumption and data quota usage. Importantly, service providers and users can take advantage of these features without requiring any support from carriers or new OS-specific apps installed by users.

Last, *Meddle* facilitates participation from a large number of users, allowing us to characterize and evaluate mobile systems in the wild. It offers a low barrier to adoption – on modern mobile OSes, users need only a few taps to install *Meddle* and once installed there is no maintenance required. Also, *Meddle* explicitly aligns the goals of researchers and users

by offering free and easy-to-use services that include device-wide ad-blocking, privacy/security filters and parental controls at the network layer – functionality that mobile network providers and OSes do not currently make available.

This paper provides the following key contributions. First, we describe the design and implementation of *Meddle* (Sec. 2). We demonstrate the scalability and reasonable overheads of tunneling traffic from large numbers of mobile users through our servers (Sec. 3). Second, we present measurement results gathered from our initial deployment of *Meddle* comprising of 14 users interacting with the system for 30 days (Sec. 4). We use data collected from our IRB-approved study to highlight key advantages that our platform offers compared to previous work. Third, we describe several services that we built on top of *Meddle* (Sec. 5). Last, we must address the additional security, trust and privacy concerns that arise when tunneling traffic outside of carrier networks into a third-party distributed service. We discuss these issues and others in Section 6. We conclude in Section 8.

2. GOALS AND ARCHITECTURE

In this section, we discuss the motivation for *Meddle*, describe our goals for the system and present an architecture to meet those goals.

2.1 Motivation

Meddle is motivated by the fact that users and researchers currently have a limited view – if any – into how devices and apps generate network traffic. In addition, there are few mechanisms available for shaping, modifying or blocking this network traffic.

This can be harmful to users, as apps may generate sufficient traffic to cause overage charges on cell phone bills. Even when the volume of traffic is small, a periodic network activity can quickly deplete battery charge by preventing power management policy of the device to correctly operate [18]. Further, apps can use the network to leak personal information or track users without consent [12].

Researchers likewise suffer from a lack of visibility and control over network traffic. It is common to use testbeds or proprietary ISP datasets to reveal network usage and its impact on data quota, battery life and privacy. However, the coverage and representativeness of these results can by limited by the use of i) a single carrier's network, ii) a small testbed with a single, custom operating system and iii) a single access technology (cellular or WiFi). Once researchers identify a problem with network traffic, addressing it can be challenging. OS developers are slow to adopt changes, app developers may be unwilling to modify their code and carriers often have no incentive to deploy new in-network functionality.

2.2 System Goals

The two primary goals for *Meddle* are 1) to provide comprehensive visibility into mobile networking traffic and 2) facilitate the development of new solutions and services for mobile networks. We further identify the following subgoals that address limitations of previous work.

- *Portability.* We want a solution that works regardless of operating system, access technology and provider, without requiring advanced (and possibly warranty-voiding) techniques such as rooting a phone.
- *Pervasiveness.* 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 devices.
- *Deployability.* Our solution should be easy to use, immediately deployable and incur reasonable costs (or none at all) for users.
- *Isolation.* To prevent existing in-network devices from interfering with these new solutions and services, our system should provide network-traffic isolation.
- *Control.* To enable new solutions and services, our system should enable users and researchers to shape, modify or block network traffic.

2.3 Design and Architecture

2.3.1 Design

To meet the above goals, we propose building a system that redirects all mobile-device network traffic to a server outside the carrier's network, thus providing a point of control where one can characterize, modify or block this traffic before sending it to the intended destination. Importantly, we observe that we can do this today without any additional support for devices or carriers: the key idea is to combine software middleboxes (e.g., packet filters, proxies, etc.) with virtual private networks (VPNs).

We use VPNs to achieve our goals of portability, pervasiveness, deployability and isolation. Specifically, major mobile OSes provide built-in VPN functionality for enterprise customers to enable access to resources in the enterprise's private network for employees "on the road". In *Meddle*, we use VPNs as a portable mechanism[1] to tunnel traffic from mobile devices to a machine outside of the carrier's network for the purpose of analysis and interposition.

We use software-middlebox techniques to meet the goal of providing control over network traffic in the mobile environment [22]. Middleboxes are traditionally used in managed networks (e.g., in enterprises and ISPs) to implement policies and enhanced services over IP. In *Meddle*, we use middleboxes as a mechanism not only to implement custom policies and services for users and service providers, but also

[1]Android, BlackBerry and iOS and Windows 8 all support VPNs natively, representing more than 86% of the mobile device market[8].

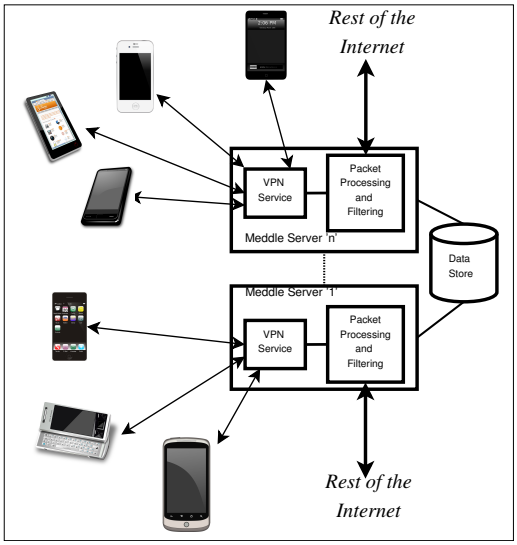

Figure 1: *Meddle* architecture. Devices use VPN connections to tunnel all traffic to one of potentially many *Meddle* servers dynamically chosen to optimize for performance. Each *Meddle* server uses a device- or user-specific profile to determine the set of services that will operate on the tunnel's network traffic.

for measuring networks and experimenting with alternative protocols for the mobile environment without requiring access to mobile carrier networks.

2.3.2 Architecture

Figure 1 depicts an architectural diagram for how we realize this approach via *Meddle*. Devices use VPN connections to tunnel all traffic to one of potentially many *Meddle* servers. We envision that *Meddle* can run as a single-user system in a user's home network or in a hosted data center, or it can run as a shared, distributed and/or virtualized cloud-based service (*e.g.*, on Amazon EC2). The figure depicts the latter, where multiple *Meddle* replicas are available for any given device.

In the distributed scenario, when a device attempts to connect to *Meddle*, we direct it to a nearby *Meddle* server in a similar way to how the Akamai CDN uses DNS to redirect Web clients to nearby content caches [3]. To optimize for performance, the service can send VPN clients to a server that is relatively near the location at which data exits the mobile carrier's network to enter the Internet. This technique can also redirect clients based on server load, reachability and availability.

To meet the goal of providing control over network traffic in the mobile environment, each *Meddle* server uses software-middlebox techniques, or *meddleboxes* to interpose on network traffic. These meddleboxes can (i) implement custom services for users such as packet filtering, caching, and intrusion detection, (ii) monitor network traffic characteristics, and (iii) allow us to experiment with alternative algorithms and protocols for the mobile environment. The system maintains a per-device and per-user profile to determine the set of services that will operate on the tunnel's network traffic.

Meddleboxes are monolithic in our current implementation, meaning that all meddlebox features are within the same *Meddle* server. When deployed in the cloud and at scale, this could lead to suboptimal scaling because users might adopt meddlebox features unevenly. To address this, we are developing a cloud-based distributed meddelbox deployment where each service runs in its own virtual machine (VM) and a user's traffic is routed through a series of these VMs using a software-defined network to set up per-user routes on demand.

While the *Meddle* architecture is straightforward, there are a number of challenges we must address in building a viable deployment. In the next section, we describe how we achieve continuous monitoring of network traffic across platforms, then demonstrate that the power, data-volume and latency overheads are reasonably small.

3. IMPLEMENTATION

In this section, we describe our current deployment, we discuss how we addressed several challenges toward achieving the goals listed in the previous section and we present empirical results demonstrating the feasibility of our approach in terms of reasonably low overhead.

3.1 Meddle Details

The design of *Meddle* facilitates widespread adoption because it is supported out-of-the-box by the vast majority of smart mobile devices (smartphones and tablets) and is easy to deploy on servers. 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 required to establish the VPN tunnels. The two most popular key exchange algorithms are IKEv1 [11] and IKEv2 [13]. Android devices support IKEv2 while iOS devices currently support only IKEv1 for tunnel establishment. The advantage of IKEv2 is that the VPN tunnel is generally established using 4 packets compared to 15 packets exchanged to establish the tunnel using IKEv1.

iPhone support. We support *Meddle* on iOS using the "VPN On-Demand" feature, introduced in version 3.0 of iOS. This was originally intended to allow enterprises to ensure their employees' devices always establish a VPN connection before contacting 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 each alphanumeric character as the set of domains that require a VPN connection. This ensures that a connection is established before *any* network activity.

Android support. As of Android 4.2, Android supports "always on" VPN connections that ensure all traffic is always tunneled. At the time of writing, this has been released only

for one week and only for a subset of Android devices so we have not extensively tested the effectiveness of the solution.

Instead, we rely on a feature that allows apps to manage VPN connections, introduced in Android 4.0. We modified the StrongSwan implementation of a VPN client [26] to ensure that the VPN reconnects each time the preferred network changes (*e.g.*, when a device switches from cellular to Wi-Fi).

Server-side implementation. In our current implementation, *Meddle* uses native IPsec, implemented via StrongSwan [25], to establish VPN tunnels and custom packet filters plus Vyatta middlebox software [**?**] to shape traffic. Both of these software artifacts are supported on vanilla Linux operating systems, which in turn run on nearly all servers.

3.2 Feasibility

Because our approach in part depends on users installing a VPN configuration and tunneling all traffic through *Meddle*, we evaluate whether the cost to the user in terms of performance, power and data quota is sufficiently low.

Data consumption. IIPsec 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 8–12.6%.

For our measurements, we capture the encrypted packets exchanged by our *Meddle* servers and the clients that use *Meddle*. We performed the packet capture for 14 days during which 20 devices tunneled their traffic via our *Meddle* 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 8.23%, with a maximum increase of 12.6%.

Performance. By forcing user traffic to an intermediate server and interposing on flows, we may add latency both due to additional hops and due to processing time at the *Meddle* server. We envision a DONAR-style deployment where users are dynamically redirected to different *Meddle* servers based on network conditions and server load [31]. Given this model, we evaluate whether we can locate servers near mobile-network egress points using a deployment such as PlanetLab, and found that this is generally the case.

For this experiment, we used data from approximately 10 mobile phones located throughout the US and issued traceroutes from the devices to targets in Google and Facebook's networks. We then used the first non-private IP address seen from the mobile device on the path to a server. We assume that this corresponds to the first router adjacent to the mobile carrier's public Internet egress point. Note that we could not simply ping the device IPs because mobile carriers filter inbound ping requests. Using this set of egress adjacencies, we determined the round-trip time from each PlanetLab site, then took the average of the nearest five sites to represent the case where a host at the nearest site is unavailable due to load or other issues. The average latency to each router was between 3 ms and 13 ms, with a median of 5 ms. Thus, when compared to RTTs of 10s or 100s of milliseconds that exist in mobile networks, the additional latencies from traversing *Meddle* servers is expected to be relatively small or even negligible.

[**TBD: What is the cost in terms of end-to-end performance?**]

[**TBD: Are there cases where we actually improve performance due to detouring? (I've decided not to mention this because it's not a sure thing.)**] [**TBD: In case the carrier is performing traffic differentiation to penalize some bandwidth consuming traffic, using a VPN might significantly improve performance. I don't know whether traffic differentiation is something common on mobile networks. (that was an answer to Dave question. If he discards it, also drop my comment.)**]

Power consumption. Tunneling traffic to a *Meddle* server requires that all traffic be encrypted by the mobile device. While this is already commonly performed for SSL connections, *Meddle* requires an *additional* layer of encryption. We observed a 10% increase in power consumption when streaming an HD video to Android and iPhone devices using our IPsec tunnel. We believe that this overhead is reasonably low, and we note that this cost for encryption comes with the added benefit of increased privacy from third parties. To put the overhead in context, the iPhone 5 advertises 8 hours of LTE browsing per charge; with the VPN enabled this would reduce to 7 hours and 12 minutes.

An interesting research question is whether it is possible to *reduce* power consumption using *Meddle*. For example, Qian et al [20, 19, 18] found that traffic shaping (a service that *Meddle* provides) can significantly reduce the power consumed by devices when periodic application traffic and radio resource timers are out of sync.

Scalability. If wildly successful, we would like to ensure that *Meddle* scales gracefully and that there are sufficient resources to support large numbers of concurrent users worldwide. Based on our initial analysis using StrongSwan on commodity hardware, we found that each connection consumed on average less than 1% of CPU time. Thus, we expect to be able to support up to hundreds or small number of thousands of users per server, which is in line with low-end VPN appliances sold by Cisco and Vyatta. A recent study [2] showed that current rates for 3G networks in the US were between 0.59 and 3.84 Mbps; assuming devices are uniformly distributed across carriers, we expect to be able to support 250 simultaneous users (saturating their download capacity) for every 1 Gbps of bandwidth at the server.

3.3 Current Deployment

The analysis in the following sections relies on data gath-

Protocol	Service	Android		iOS	
		Flows	Bytes	Flows	Bytes
TCP	HTTP	14.87	74.31	16.07	82.01
TCP	SSL	37.19	24.35	37.64	17.31
UDP	-	40.52	0.98	42.34	0.49
TCP	other	2.08	0.22	1.13	1.89
Other	-	5.35	0.14	2.82	0.09
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.*

Protocol	Service	Wi-Fi		Cellular	
		Flows	Bytes	Flows	Bytes
TCP	HTTP	16.91	83.52	13.03	56.14
TCP	SSL	38.18	15.83	36.14	41.42
UDP	-	41.43	0.49	41.56	1.67
TCP	other	1.38	0.16	1.95	0.4
Other	-	2.12	0.01	7.2	0.3
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.*

ered from our current prototype deployment. This consists of *Meddle* servers at the University of Washington, UC Berkeley and Inria, along with 20 devices from 14 users participating in an IRB-approved study.

Summary statistics. *Meddle* has been running since September 2012. The deployment includes 8 Android devices and 12 iOS devices (4 iPads, 7 iPhones and 1 iPod Touch). These devices connect to *Meddle* via 15 different service providers, four of which are cellular providers. Together, our users have generated 30.2 GB of traffic since November 1, 2012.

Data collected. We collect full packet traces from these users (with informed written consent) and we code the data using random identifiers to protect identities. We do not attempt to decrypt any SSL traffic, and all packet traces are encrypted using a public key before being stored on disk. The private decryption key is stored on a separate server. Users may opt out and choose to delete their data at any time.

4. CHARACTERIZING MOBILE TRAFFIC

4.1 Descriptive Statistics

In this section, we use several descriptive statistics to summarize our dataset and provide context for the remainder of the section. We focus on the type of traffic being generated, how it varies per OS and how it varies over different access technologies.

Comparing across OSes. A key advantage of our approach is that it allows passive network monitoring across multiple OSes. In contrast, studies that rely on a custom, modified OS can monitor only that OS. Table 1 examines the portion of traffic using several key protocols for iOS and Android devices.

We classify IP flows as either TCP, UDP, or other. We further classify TCP flows as either HTTP, SSL, or other. The SSL flows include HTTPS, IMAP, and other services that rely on SSL to secure the connection between the mobile client and the remote server. TCP flows that are not classified as either HTTP or SSL are classified as other.

First, we observe that DNS is responsible for more 93.5% of the UDP flows. The rest of the flows are due to services such as Skype.

Second, we observe that most TCP traffic traffic is HTTP(S), in line with previous results. Third, the majority of byes are transferred over unencrypted channels. This is mainly due to streaming video (*e.g.*, Netflix and YouTube) over HTTP.

Last, we find the majority of TCP flows are sent over SSL, and these fraction is nearly identical for Android and 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 channel regardless of the OS.

On balance the number of bytes transferred per unit time in Android, and the number of flows, is larger than the same for iOS. This can be explained by 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. [**TBD: @Ashwin: We need stats**]

Comparing across access technologies. Another key advantage of *Meddle* is that it allows us to passively monitor traffic regardless of whether a device is on Wi-Fi or cellular, without needing any OS modifications or imposing any constraints on user mobility. In contrast, studies relying on traffic seen over campus Wi-Fi [4] miss traffic over cellular connections; those from cellular providers miss Wi-Fi traffic. As a point of reference, we found that the portion of network flows generated by devices were nearly equally split (60/40) between Wi-Fi and cellular.

In Table 2, we summarize the network traffic generated by *Meddle*-enabled devices over Wi-Fi and cellular. 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. Another key reason is that apps such as Google Plus (image backup on Android) allow users to upload their images only over Wi-Fi.

The portion of "other" flows for cellular is a relatively large 7.2%. This occurs because several users run *Mobiperf*, an application that performs regular network-measurement

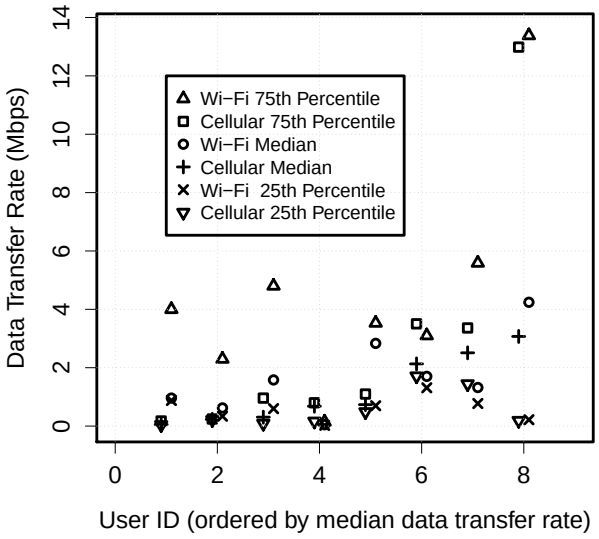

Figure 2: Median and quartile peak transfer rates for users when on Wi-Fi compared to cellular. Many users see significantly better performance from Wi-Fi; however, users with LTE often receive better performance over cellular than Wi-Fi.

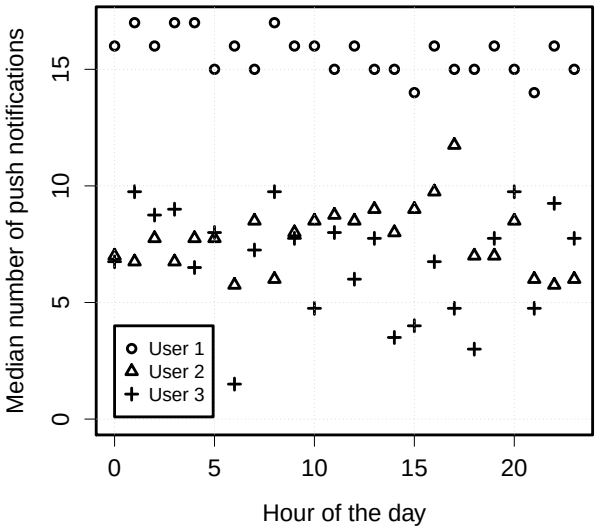

Figure 3: Median number of push notifications seen per hour for three devices.

tests only over cellular connections.

Enabling direct performance comparisons. Historically, cellular performance has lagged significantly behind Wi-Fi, mainly for capacity reasons. With the advent of LTE and the increased interference from dense Wi-Fi deployments, we ask whether this common knowledge remains valid.

Previous work by Sommers et al. [23] investigated this question using SpeedTest results from a metro area, but did not have sufficient location accuracy to directly compare performance over Wi-Fi and cellular for the same device. They found better performance from Wi-Fi in the vast majority of cases.

To directly compare Wi-Fi and cellular performance for the same device, we determine the maximum transfer rate per hour for each device, then focus on cases where there was a sample for Wi-Fi and cellular in the same hour. We focus on users that transferred more than 100 MB of data in a single month. Figure 2 depicts the median and quartiles of these peak rates. The graph shows that for many users (left side of the figure) Wi-Fi is better than cellular. However, users on the right side often have better peak transfer rates over cellular – mainly due to adoption of LTE. For user 4, however, we see that Wi-Fi performance is poor compared to other users, indicating that users may see better connections over cellular connections even if they are 3G or older.

4.2 Case Study: Device Specific Apps (iOS Push)

In this section, we present the results of a detailed case study of iOS push. We focus on this application because it represents an OS-managed service on an operating system that received little research attention due to its closed-source nature.

We set out to answer the following questions: (1) What is the empirical network impact of push notifications and does the measured activity coincide with published documentation? (2) Using *Meddle*, is there any relevant data traffic that is not captured by the VPN tunnel? (3) What is the impact of a VPN connection on network behavior? (4) Can we use network traffic alone to infer the state of the device?

iOS Push Under the Microscope. 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.

We now further investigated the behavior of iOS push over time for three characteristic users. Figure 3 plots this in terms of median number of push notifications received per user per hour. During the period, all users were in the same time zone.

First, the figure shows that all three users see significant numbers of push notifications even in idle, overnight periods. It is unclear what the value of such traffic is. Second,

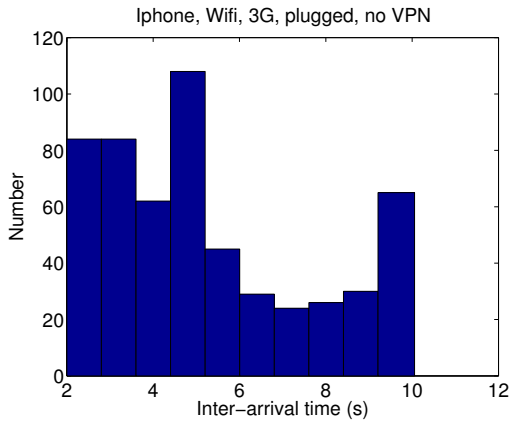

Figure 4: Distribution of the interarrival times of Ethernet frames for a one hour experiment with an idle iPhone plugged-in, with Wi-Fi and cellular enabled, and no VPN enabled. For each bin of 1 second, we count the number of interarrivals in that bin.

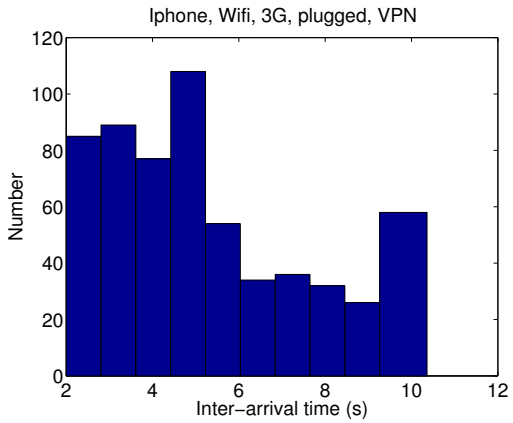

Figure 5: Distribution of the interarrival times of Ethernet frames for a one hour experiment with an idle iPhone plugged-in, with Wi-Fi and cellular enabled, and VPN enabled. For each bin of 1 second, we count the number of interarrivals in that bin.

we clearly see significant differences in the volume of traffic among users. User 1 has registered to receive push notifications from multiple apps, while Users 2 and 3 have not. Last, we note that even with iOS 6's "do not disturb" (DND) feature, push notifications continue to be sent during DND periods. Specifically, User 2 has enabled DND between midnight and 7am, but there is no noticeable change in APNS traffic. In summary, we can use *Meddle* to help understand the network behavior of closed systems; further, we identify several opportunities where we believe the OS can reduce or eliminate push traffic (during idle periods).

Interaction between *Meddle* and the power management policy on iOS. In the remainder of this section, we perform controlled experiments to understand the interaction between *Meddle*'s VPN connection and the power management policy on iOS. In summary, we find that the impact of the VPN depends on the power state and network interfaces used (Wi-Fi or cellular) on the device – but we show that the current power management policy on iOS might adversely impact the energy consumption when the APNS is heavily used. Further, we find that the changes in traffic patterns are relatively easy to identify, meaning *we can use traffic from* Meddle *to infer when a device is plugged in and which access technologies are available*.

All the following experiments are performed on an iPhone 4 with iOS 6.0.1. At the beginning of each experiment we close all applications and restart the iPhone. The iPhone is connected in Wi-Fi to a controlled access point on which we perform a tcpdump on the Wi-Fi interface and monitor the Wi-Fi association between the access point and the iPhone. Each experiment lasts for around 1 hour.

Behavior when plugged in. When an iPhone is plugged-in, it always remains associated with the Wi-Fi access point if Wi-Fi is enabled on the device. We performed a first set of experiments to observe the traffic to and from an idle iPhone.

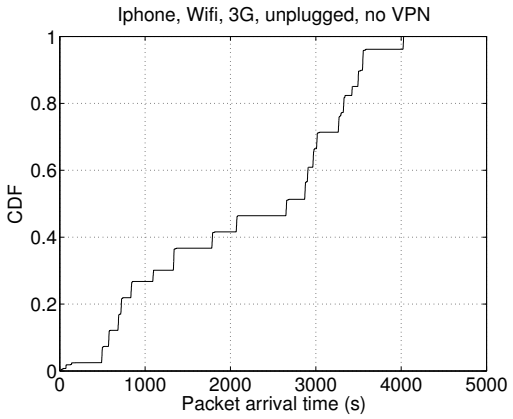

Figure 6: Cumulative distribution of the Ethernet frames arrival with time for a one hour experiment with an idle iPhone unplugged, with Wi-Fi and cellular enabled, and no VPN enabled.

We see in Fig. 4 and Fig. 5 that the largest interarrival between two frames is in the order of 10 seconds for both the VPN and no VPN scenarios. We also observe no noticeable difference in the arrival of frames with time for both scenarios. (The corresponding figures are omitted for space reasons.) *Thus, it appears that the VPN connection has no noticeable impact on network traffic pattern when the device is plugged in.*

Behavior when on battery. In a second set of experiments, we consider a default iPhone setting with cellular and Wi-Fi enabled, and the iPhone unplugged. This corresponds to a typical setting for a user on move. We observed relatively large interarrival times for frames in this case (many on the order of 100s of seconds). These cases correspond to periods during which the Wi-Fi interface of the iPhone is not associated to the access point for energy saving.

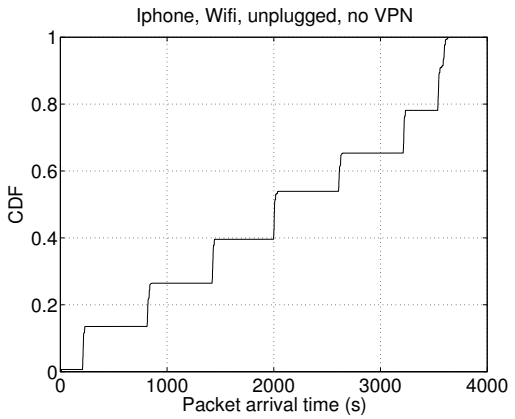

Figure 7: Cumulative distribution of the Ethernet frames arrival with time for a one hour experiment with an idle iPhone unplugged, with Wi-Fi enabled, and no VPN enabled.

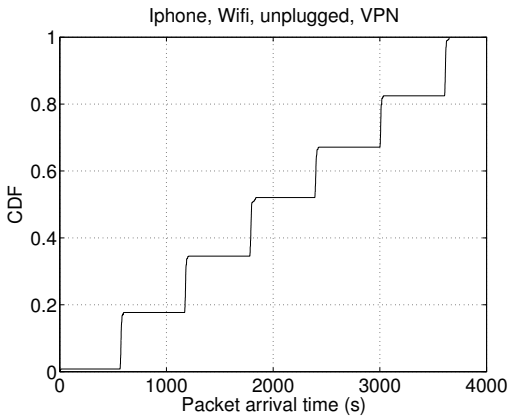

Figure 8: Cumulative distribution of the Ethernet frames arrival with time for a one hour experiment with an idle iPhone unplugged, with Wi-Fi enabled, and VPN enabled.

Surprisingly, even though the iPhone is idle, the traffic pattern is not regular (Fig. 6). To understand this irregularity, we made a new experiment with cellular disabled. The largest interarrival is still in the order of 550 seconds, as in the case with cellular enabled, but we have less variety in the interarrival times. This is confirmed by Fig. 7 that shows a regular succession of short periods during which frames are exchanged and long periods during which the wifi interface is not associated to the access point.

Therefore, there is an evidence that the cellular connection triggers the association of the iPhone Wi-Fi interface with the access point. This is confirmed by experiments we performed with iMessage, which uses the push notification infrastructure of Apple. When both cellular and Wi-Fi are enabled on an iPhone, a permanent connection to the push notification infrastructure of Apple is setup on the cellular interface. Each time an iMessage is received, it is piggybacked on the push notification message (when short enough). Interestingly, the Wi-Fi interface is always woken-up even if no payload is sent on it. We don't know the exact reason for this behavior, but it may have an adverse impact of battery life for users with frequent push notifications, *e.g.*, heavy users of SMS, twitter, facebook, games, *etc.*

In summary, APNS allows apps to interact with the user despite a strict energy-saving policy that prevents apps running in the background; however, *the systematic wake up of the wifi interface might lead to an unnecessary severe increase in power consumption*.

Behavior when Meddle *is enabled.* We performed a third set of experiments with the VPN enabled. This set of experiments shows the interaction between *Meddle* and the power management policy. We see in Fig. 8 that there is no noticeable differences when enabling *Meddle* with Wi-Fi only. However, when enabling both Wi-Fi and cellular the frames interarrival time and the cumulative arrival of frames (Fig. 9) is significantly different. This difference between scenarios with and without cellular is due to the persistence of the cellular connection. Indeed, on the one hand, if messages are sent to the device when the cellular connection is disabled, they will be lost if the Wi-Fi interface is sleeping. Therefore, the Wi-Fi association periods are triggered by an internal device policy only. On the other hand, when the cellular connection is enabled, all messages sent to the device will be received. If the Wi-Fi interface is sleeping, messages received on the cellular interface will wake up the Wi-Fi interface. In that case, the Wi-Fi association periods are not only triggered by an internal device policy, but by external events too, which explains the very different interarrival times seen in Fig. 6 and Fig. 7, and in Fig. 8 and Fig. 9.

The main question that remains is: what are these messages received on the cellular interface? In Fig. 9, we observe a pattern with a periodicity of around 45 seconds. This pattern does not exist when the VPN is disabled as shown in Fig. 6. We conclude that this periodicity is mostly due to keep alive messages of the VPN for NAT traversal. In Fig. 6, we observe what looks like a random wake up of the Wi-Fi interface when the VPN is disabled. We explored tcpdump traces for this scenario and for the scenario with the VPN enabled, looking for messages that explain the apparent randomness, but we did not find any such messages. Therefore, we guess that packets that do not use the VPN tunnel are received on the cellular interface. Such packets might be sent by the cellular provider to the device.

In conclusion, though *Meddle* appears to capture all IP traffic it may miss traffic in the circuit switched network when a cellular connection is enabled.

4.3 Case Study: Longitudinal Measurements - Google Search

A key advantage of our approach is that we can monitor app behavior over time and as updates occur. This can lead to surprising or otherwise unpublicized revelations about network activity.

One prominent example is Google searches from mobile

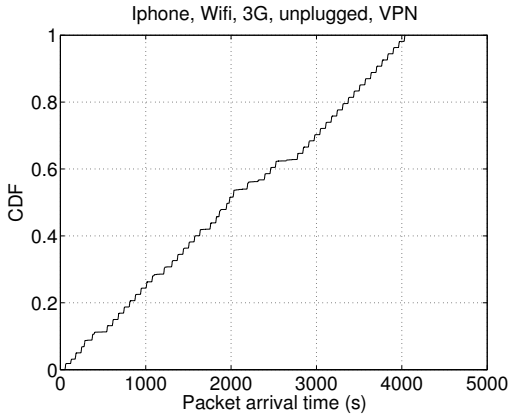

Figure 9: Cumulative distribution of the Ethernet frames arrival with time for a one hour experiment with an idle iPhone unplugged, with Wi-Fi and cellular enabled, and VPN enabled.

devices. In the desktop environment, we have grown accustomed to these searches occurring over HTTPS connections, which protects user privacy from carriers and from other users at open Wi-Fi hotspots.

In the mobile environment, however, we noticed that the default browsers on iOS and Android send user queries in the clear. In fact, each letter is sent as the user types it for the purpose of Google Suggestions. We observed this behavior on Android 4.0, Android 4.1, and Android 4.2 [**TBD: confirm**]. In the case of iOS devices we observed search queries in the clear on devices running iOS 5.

Interestingly, as of iOS 6 these searches are now sent using HTTPS, addressing a significant privacy vulnerability. To the best of our knowledge this change has not been publicized. Surprisingly, Google's own operating system's default browser has not made the change to HTTPS for searches.

4.4 Case Study: Misbehaving Apps and Filtering

Smartphone users are often faced with the decision whether or not to trust an app downloaded from the iPhone App Store or the Android Market. While the user can access some information a priori regarding the app's behavior, *e.g.* via reading reviews or inspecting the permissions requested,they ultimately can not know the app's behavior without installing it. While some undesirable app behavior is readily apparent – *e.g.* pushing advertisements to the notifications bar, installing unwanted third-party software, *etc.* – the app's network behavior remains invisible to the user even after installation. With this in mind, we used *Meddle* to perform a short app-by-app analysis of network usage behavior.

To start, we defined the network usage properties we desired from a 'well behaved' app:

- Protection of personally identifiable information (PII): transmitting PII only when required for application functionality, and if transmitting PII, doing so over HTTPS rather than HTTP.
- Conservation of power and data quota by making only necessary use of network.
- Communication only with servers needed for the applications intended purpose, with preferably a limited number of entities contacted by the app overall.

We then downloaded six of the most popular apps from the Android Market, installed them on an fresh install of Android 4.1. One was a popular game, one was the app for a widely used image sharing site, one was a 'weather' app, one was a settings management app, one was the app for a popular movie, and the final was a risqué app serving various images.

The movie app, the settings app, and the game app easily fit our provider. Over 30 minutes of active use with the movie app, the phone contacted Amazon, Google, and a single CDN provider; the median inter-packet transmission time was [**TBD: x**]. Over 30 minutes of active use with the settings app, the phone contacted Amazon, Google, a hosting service, and two CDN services; the median inter-packet transmission time was [**TBD: y**]. Over 30 minutes of active use with the game app, the phone contacted only Google and Amazon; while the median inter-packet transmision time was higher ([**TBD: z**]), we presume this was needed for updating the game scoreboard at the end of each game.[**TBD: Amy, is this a reasonable estimate?**]

The image-sharing and weather app we classified as likely non-malicious. They did not leak any PII in their traffic traces, however, they did violate our expectations of well-behaved apps. The risué app we classified as malicious, however its network behavior only violated our expectations in one dimension. Time plots of network traffic for these three apps are shown in Figure **??**.

The image-sharing service violated our definition of a well-behaved app by contacting too many entities and generating too much trafic. During 30 minutes of use, the phone contacted 10 CDNs, exposing the user's browsing behavior to a large number of organizations. Further, the inter-packet arrival time was a median of [**TBD: alpha**], resulting in sustained network usage and potentially impacting battery life.

The weather app similarly contacted a large number of CDNs and organizations, for a total of 10 organizations contacted. Not only does this network footprint raise concerns about exposure to numerous entities, but furthermore, one of the organizations contacted explicitly serves as a tracking service; their homepage advertises universal tracking for smartphone devices. Although they advertise this service for fraud prevention, it is hard to imagine what fraud prevention is necessary for a simple weather app.

Finally, the risqué app we did classify as malicious; it pushed advertisements to the notification bar and installed unwanted extra apps. However, its network profile was fairly benign. We observed no leaked PII. However, it did contact 19 entities including 7 CDNs and 4 advertising networks.

This illustrates that monitoring network traffic is only part if the arsenal in monitoring app behavior.

All three of our borderline apps contacted multiple ad network, thus a potential application of *Meddle*, which we discuss in the following section, is to simply filter all traffic to ad networks. To verify that blocking ad networks would not interfere with normal app behavior, we enabled filtering on the ad networks that the apps contacted, and manually tested each app's behavior... [**TBD: more tomorrow.**]

4.5 Case Study: Traffic Optimization - Compression

[**TBD: Ashwin: point out cases where compression hurts. also show how much is already compressed.**]

5. CONTROLLING MOBILE TRAFFIC

In this section, we use two case studies to highlight some of the advantages *Meddle* provides as a new point of control for mobile networking traffic. We then discuss several new applications we are currently building.

5.1 Packet Filtering

Packet filtering is a common technique implemented in today's middleboxes. These can be used for implementing security policies, censoring content or preventing applications from harming the network (*e.g.*, P2P). In *Meddle*, we use packet filtering to enable a number of features that are currently either unavailable or poorly implemented.

Device-wide ad blocking. Content and service providers typically use revenue from advertisements to cover their costs, enabling "free" access for users. Given the additional costs in terms of data volume, battery consumption and potentially reduced privacy from tracking, it is unclear just how "free" these services truly are. Vallina-Rodriguez *et al.* [28] observe that ads account for 5% of daily traffic from more than 50% of Android users in a large European ISP.

To understand the impact of ads using *Meddle* as a vantage point, we analyzed our traces and identified traffic for ads and analytics (A&A) servers in terms of the portion of total traffic. Figure 10 presents our results. The x-axis represents users and the y-axis indicates the percent of total traffic generated by A&A servers for all traffic and for only cellular traffic.

We first observe that the amount of A&A traffic is highly variable but can be significant. For example, user 1 sees approximately 54.25 MB of ad traffic while user 11 sees approximately 3% of cellular traffic coming from A&A servers, which corresponds to 8.31 MB.

Next, we observe that the amount of ad traffic can differ significantly based on whether the user is on cellular or not. Thus, a view of A&A traffic that considers only cellular traffic can miss significant activity. To emphasize this point, we include the following ratio in parentheses for each user: (Volume of Cellular Ad Traffic / Volume of Wi-Fi Ad Traffic) / (Volume of All Cellular Traffic/ Volume of All Wifi-Traffic) This ratio gives an indication if ad traffic differs across access technology. Users that do not have a cellular connection (tablets) have a value (-). Most A&A traffic occurs relatively evenly regardless of access technology (most of users have value in the range of 1). For user 4, there is a clear exception. We were able to track this to a specific app using the request user agent. In this case, the we identified the traffic as coming from the Field Trip app. [**TBD: This is probably actually coming from Ingress, a very addictive "enhanced reality" game in private beta. Not sure how to talk about this.**]

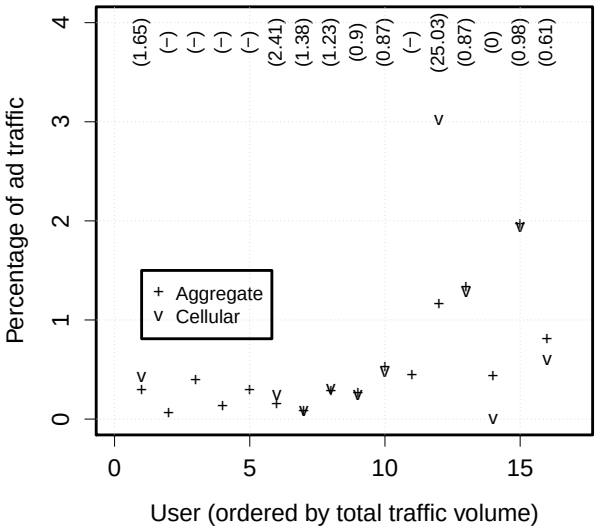

Figure 10: Portion of user's traffic generated by ads and analytics (A&A). A significant portion of traffic (up to 4%) comes from A&A servers. For most users the A&A traffic occurs independently of access technology, but for some users it occurs much more often on cellular than on Wi-Fi.

Our results, along with related work, highlight the fact that A&A traffic can be significant. However, there is no current standard for opting out of unsolicited advertising; those that exist for tracking are not widely supported. Not surprisingly, many users have turned to software that blocks ad and analytic servers. For example, more than 20 million users have installed AdBlockPlus [2], one of several tools for this purpose. Unlike the desktop environment, mobile device browsers do not provide support for ad blocking; further, most interactions occur inside of apps where browser plugins cannot help.

Meddle makes it easy to implement an efficient, device-wide ad blocker. In our current implementation, we use a DNS-based filter to block ads, analytics, and mediation sites.[3] A key feature of our solutions is that it works regard-

[2]http://www.adblockplus.org

[3]The service is disabled by default, so users must opt in to enable

less of whether SSL is used because DNS requests occur out of band from the secure connection. Further, the response from the DNS request is `localhost`, meaning that devices will generate no external network traffic when failing to resolve the ad servers.

Our ad blocking engine relies on a publicly available list of domains for ads and analytics [1]; we augment this list of domains using recent research on mobile ads [12, 15]. From our initial deployment, we observed a 0.05% to 4% reduction in total traffic at each mobile device due to our ad blocking engine. In addition to the DNS-based filter, we are currently implementing a filter that blocks requests for URLs matching a regular expression, as done by AdBlockPlus.

[**TBD: Mention Privad gateway**]

Do not disturb. Many users do not turn off their devices at night, even while asleep. In our study, we found that [**TBD: XX**]% of devices were active for at least one entire 24-hour period. Network traffic occurring during those periods is generally not helpful for the user; worse, such traffic may cause audible alerts that wake up the user. In part to address this, iOS 6 introduced a "Do not disturb" feature that silences the phone and *reduces* network activity. We are unaware of such support on Android.

Meddle enables a cross-platform and cross-device "Do not disturb" for network traffic. Users specify the hours when traffic should be blocked and the corresponding meddlebox will drop all traffic. Further, users can specify hosts that should *not* be blocked.

Parental controls. Existing parental controls for mobile devices tend to focus on which apps a user cannot run. For example, a parent can block the Facebook app on an iOS device. To the best of our knowledge, there is no way to prevent a user from using a Web browser to access the Facebook site, unless all Web browser activity is blocked.

In general, this would be too prohibitive. Using *Meddle*, we can blacklist traffic for all IPs known to belong to a content or service provider. For example, to block Facebook we would block IPs belonging to Facebook, as well as any traffic using Facebook SSL certificates (*e.g.*, for accessing content hosted by CDNs).

5.2 Traffic Manipulation

An advantage of *Meddle* is that it provides isolation from carriers' in-network middleboxes, which may block, modify or shape certain network traffic. However, toward the goal of improving transparency in mobile networks (especially for non-*Meddle* users), we would like to be able to identify when, how and what network traffic is subject to carrier manipulation.

In this section, we describe how we use *Meddle* to detect HTTP interference via in-flight Web page changes. We believe our technique is general to other protocols.

The problem of HTTP interference was highlighted by Reis et al. [21]. The authors demonstrated that although a small precent of users were affected by in-flight changes, those changes tended to introduce vulnerabilities such as overflows and cross-site scripting (XSS) attacks. They proposed and deployed *Web Tripwires*, Javascript code that detects in-flight page changes. The main limitation of this study is that the approach works only for sites that include a tripwire on their own pages.

blocking.

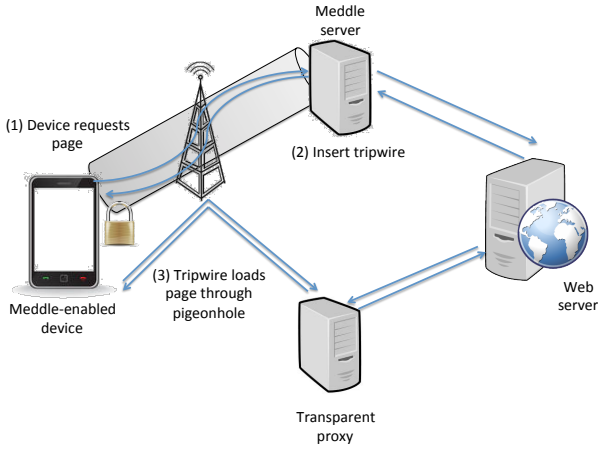

Figure 11: Overview of the *Meddle* Web tripnet. A Web page is loaded through the VPN tunnel, where a meddlebox inserts a Web tripwire. The tripwire causes the browser to reload the Web page using the address of a transparent proxy server that is accessed using an unencrypted connection. After the proxied version of the page is loaded, a message is sent to the user if the pages differ due to ISP interference.

Web Tripnets. Using *Meddle*, we extended tripwires to alleviate this limitation. Namely, we use *Meddle* to inject a tripwire on *any* page without requiring support from Web site developers – an approach we call a *Web Tripnet*.

When enabled, the Web tripnet works as follows (depicted in Fig. 11). First, the device requests a Web page via HTTP (1). Because the *Meddle* connection is encrypted, the device's provider cannot alter the page in flight. When the Web server response arrives at the *Meddle* server (2), we insert a Web tripwire before forwarding the response to the user.

To detect whether the carrier would have manipulated the page without *Meddle*, we create a *pigeonhole* by configuring the device's VPN settings to send traffic for a particular site in the clear. When the Web tripwire attempts to load another copy of the Web page (3), the request is redirected (using DNS) to a transparent proxy that forwards the request to the intended server, retrieves the Web page and delivers it to the device in the clear (thus exposing it to carrier manipulation). Upon receiving the page over an unencrypted channel, the device's browser sends both versions of the Web page (tunneled and untunneled) to the *Meddle* server for analysis.

Experimental Results. We conducted a survey of ISP HTTP injection using two US carriers (Verizon and T-Mobile) and

one French carrier (Orange). To determine if the user agent can impact manipulation from carriers, we tested using Android, iOS and a desktop (tethered) browser. We tested the top 100 sites according to Alexa, in addition to a random sample of 100 of the 10,000 most popular sites. Our analysis did not identify any carrier manipulation. This is not entirely surprising, since most manipulation identified by Reis et al. occurred due to software installed inside a user's network.

An interesting observation is that we found one example where Facebook over Verizon included a promotional link specific to Verizon, even when the connection was proxies through a host at the University of Washington. Further analysis showed that the Squid proxy's `X-Forwarded-For` field was being populated with an address in Verizon's network, and Facebook was modifying the page accordingly. When we removed the `X-Forwarded-For` field, the promotional link no longer appeared.

5.3 New Applications

Meddle allows researchers to explore the space of applications that offload networking traffic from mobile devices to server where power and data quota are less scarce. We envision several important applications that can benefit from our approach.

- **P2P offloading.** P2P services tend to benefit from multiple connections and symmetric bandwidth contributions from participating hosts. Both of these are costly in the mobile environment. *Meddle* can ease these costs by offloading the logic for maintaining P2P connections to a meddlebox.
- **Privad.** Instead of simply blocking ads, *Meddle* is an ideal location to deploy anonymizing systems that support targeted advertising, as suggested by Guha et al. [10].
- **Tor.** *Meddle* improves privacy in mobile networking by encrypting the connection from the mobile device to the *Meddle* server. We can further improve privacy and anonymity by providing a Tor meddlebox that establishes and manages onion-routed connections.
- **Privacy Preserving Photo Sharing.** [TBD: Add text]

6. DISCUSSION

In this section, we discuss several key issues and limitations for an indirection-based deployment such as *Meddle*.

Privacy and trust. As discussed in Section 3.3, we collect traces from users as part of an IRB-approved study. We take great care in protecting user privacy – data is encrypted before being stored, the private key is not stored on the server where data is recorded and any PII accidentally sent in the clear is stripped from our datasets as soon as we identify it.

While this fine-grained data is useful for informing the design of meddlebox solutions (*e.g.*, identifying and stripping personally identifiable information from unencrypted HTTP traffic), it can be prohibitive for a large-scale study. In the next phase of our deployment (currently under IRB review) we will capture only packet headers and lengths. With a lower privacy risk, we believe we can recruit a larger number of users and obtain informed consent via a Web site.

Regardless, users may still be uncomfortable with sending all their traffic through a *Meddle* service, be it in a hosting center or in the cloud. We are making all of our code open source so that users can install *Meddle* on servers in their own networks (*e.g.*, in their home network). Users may opt to use this option instead; however, they will also be responsible for updating *Meddle* to include the latest new features and bug fixes. We expect that most users interested in running *Meddle* will be content with using our hosted *Meddle* service with anonymized packet headers being collected.

Acceptable use. Similar to any service providing Internet access, *Meddle* needs an acceptable use policy (AUP) to ensure that we are not liable for user activity. We model our AUP after the one provided by EC2, one of our potential hosting providers. Users are informed of this AUP at install time. If we are notified of an AUP violation, we can isolate the user generating the traffic because each user is given separate certificate-based credentials.

ISP objections. Many mobile carriers deploy in-network middleboxes for traffic engineering purposes. By tunneling *all* traffic, these boxes lose the ability to implement ISPs' policies, which can lead to suboptimal performance for other users sharing the network.

We believe such concerns are either overblown or misguided. First, we do not expect *every* device to run *Meddle*. If we were to attract even 1% of mobile users that would be a surprisingly huge success. Thus we do not expect *Meddle* to significantly impact overall traffic in a mobile network. Second, if *Meddle* traffic were to become a significant traffic engineering challenge, we argue that such information, and the policies that address this challenge, need to be transparent to users. The current model of transparent middleboxes is not in the spirit of an open Internet.

Scalability and reliability. If successful, *Meddle* will face scalability and reliability problems as the number of users increases. We believe that, in return for valuable measurement and experimentation data, we can justify funding to support some number of thousands of users. For a larger deployment, we may have to consider a paid service or alternative economic models. By running the service in the cloud and using DNS-based redirection, we expect service unavailability to be relatively rare. Importantly, we expect that mobile connectivity in general will be less reliable than connections to *Meddle* servers.

Limitations. We believe that *Meddle* enables a large number of interesting studies and experiments in the mobile environment. However, we do not claim it is a panacea.

First, *Meddle* captures only network traffic. It does not allow us to gain access to sensors on the device, information about what apps are running or control over those apps on

the device. An interesting question is what are the limits of what one can infer from a device's network traffic alone. We have shown that iOS network traffic exhibits clear, identifiable differences when plugged in versus running on battery, lending evidence that we can successfully infer properties of the phone previously available only by querying from an app on the device. Likewise, we used SSL certificates to distinguish which app generates requests to a CDN.

Second, *Meddle* may miss some data traffic. For example, we identified that iOS push was using signaling on a circuit-switched channel (SMS). We believe the volume of "missing" traffic is small; however, it remains to be seen how this holds generally and over time. As mobile networks transition to all-IP, it will be interesting to see if an approach like *Meddle* can capture all of a device's traffic.

Last, *Meddle* incurs costs that may dissuade adoption. We discussed several of the overheads based on establishing a VPN tunnel. One area of future work is to improve VPN protocols to reduce the time to establish a connection.

7. RELATED WORK

Meddle builds upon two existing technologies: VPNs and middleboxes. In our current implementation, we rely on an IPsec [14] implementation to tunnel traffics to *Meddle* servers. Sekar et al. investigated consolidating middleboxes to reduce costs and improve load balance. Sherry et al. [22] explore the opportunities enabled by moving middleboxes to the cloud, which includes simplifying management for enterprise network administrators. In contrast, our work focuses on the mobile capabilities software middleboxes enable rather than a redirection architecture for enterprise networks.

Meddle provides researchers with a cross-platform, cross-carrier technique for capturing network usage patterns from mobile devices. Previous work used on-demand active measurements to characterize network measurements [30, 24]; however, these measurements are restricted to the point at which users run the tests. Gerber et al. [9] use passive measurements alone to estimate transfer rates in a single carrier's network; *Meddle* enables such analysis across multiple carriers.

We use *Meddle* to investigate security and privacy issues in the traffic that mobile devices generate. By monitoring and controlling information at lower layers in the software stack, previous work [7, 12, 27] has shown that existing apps leak significant private information. In *Meddle*, we take this downward mobility to an extreme, moving off the end-host entirely and eliminating the restrictions of a lab setting. We demonstrated that many identifiers are being leaked in the clear, which our approach can block or modify without needing any additional support (*e.g.*, taint tracking) from the operating system. We are investigating opportunities for detecting information leakage when apps use encrypted channels and/or obfuscation.

Meddle allows us to improve data and energy consumption by blocking unsolicited content such as advertisements, and by blocking unwanted traffic during idle periods. Vallina-Rodriguez et al. [29] demonstrated that ads account for a significant portion of data consumption. Pathak et al. [17] showed that ads can consume a significant fraction of an app's overall energy consumption. Importantly, *Meddle* not only allows us to characterize unsolicited content, but also allows us to block it without OS or app modification.

The CloneCloud [5] and MAUI [6] projects explored the space of moving functionality from mobile devices into the cloud. By interposing on user traffic, *Meddle* will allow us to explore some of this functionality without requiring device modifications.

8. CONCLUSION

We're done.

9. REFERENCES

[1] Ad blocking with ad server hostnames and ip addresses. `http://pgl.yoyo.org/adservers/`.

[2] 3G/4G performance map: Data speeds for AT&T, Sprint, T-Mobile, and Verizon. `www.pcworld.com/article/254888/3g4g_performance_map_data_speeds_for_atandt_sprint_tmobile_and_verizon.html`.

[3] Akamai. Akamai CDN. `www.akamai.com`.

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

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of Eurosys*, 2011.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proc. of MobiSys*, 2010.

[7] 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.

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

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

[10] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *Proc. of USENIX NSDI*, NSDI'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[11] P. Hoffman. Algorithms for internet key exchange version 1 (ikev1). 2005.

[12] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of CCS*, 2011.

[13] 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.

[14] S. Kent and K. Seo. Security architecture for the internet protocol, 2008.

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

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

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

[18] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Periodic transfers in mobile applications: network-wide origin, impact, and optimization. In *Proc. of WWW*, 2012.

[19] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proc. of MobiSys*, 2011.

[20] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource aloptlocation for 3G networks. In *Proc. of IMC*, 2010.

[21] 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.

[22] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud services. In *Proc. of ACM SIGCOMM*, 2012.

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

[24] Speedtest.net mobile. `www.speedtest.net/mobile.php/`.

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

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

[27] S. Thurm and Y. I. Kane. Your apps are watching you. accessed july 19, 2012. `online.wsj.com/article/SB10001424052748704694004576020083703574602.html`.

[28] N. Vallina-rodriguez, J. Shah, A. Finamore, Y. Grunenberger, H. Haddadi, K. Papagiannaki, and J. Crowcroft. Breaking for Commercials : Characterizing Mobile Advertising. *Proc. of IMC*, 2012.

[29] 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.

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

[31] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *Proc. of ACM SIGCOMM*, 2010.