2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)

The Cost of Push Notifications for Smartphones using Tor Hidden Services

Stephan A. Kollmann
Computer Laboratory
University of Cambridge
Cambridge, UK
Email: sak70@cl.cam.ac.uk

Alastair R. Beresford
Computer Laboratory
University of Cambridge
Cambridge, UK
Email: arb33@cl.cam.ac.uk

***Abstract*—Push notification services provide reliable, energy efficient, store-and-forward messaging between servers and clients. This mode of communication is widely used, and sufficiently compelling for mobile devices that push notification services are integrated into operating systems. Unfortunately, push notification services today allow the service provider to practice censorship, surveillance, and location tracking. We explore whether running a Tor hidden service from a smartphone offers a viable, privacy-aware alternative. We conduct empirical measurements in the lab as well as modelling using data from 2 014 handsets in the Device Analyzer dataset. We estimate the monthly median cost of cellular data required to support a Tor hidden service from a smartphone at 198 MiB. We further estimate that the network activity would cost at least 9.6% of total battery on a Nexus One device with a daily charging cycle and connected to the Internet via 3G. We explore four strategies for reducing cellular data costs which, when combined, could potentially reduce the total monthly median cost to 61 MiB.**

I. Introduction

Push notification services provide reliable, energy efficient, store-and-forward messaging between servers and clients. This mode of communication is sufficiently compelling for mobile devices that push notification services are integrated into operating systems. For example, Google Cloud Messaging (GCM) is embedded into Android through the Google Play Services API. GCM is also available as a library for developers of iOS apps and developers of extensions for the Chrome web browser. Consequently, push notification services are widely used by apps to support both device-to-device communication (e.g. sending and receiving messages between users of a social media app) as well as supporting information dissemination (e.g. news apps and sports score apps).

Push notifications provide app writers and client device owners with four advantages: first, if the client device is switched off, or temporarily disconnected from the Internet, the push notification service will store messages and deliver them when the device is next online; second, push notification software on the client initiates a single long-lived TCP connection from the client device to the service, avoiding issues with NAT and firewalls as well as removing the need to poll servers periodically for updates; third, multiple messages destined for a variety of apps on a single client device can be coalesced temporally and multiplexed down a single TCP connection, saving battery life and improving performance; finally, an app server can achieve *service fan-out* by sending a single copy of a message to a push notification service and requesting that the message is delivered to many devices on a group or topic basis.

There are downsides to push notifications however. From a privacy perspective, a push notification service has the disadvantage that the service can see the sender and the recipient of every notification across a broad range of apps and thus may conduct surveillance and censorship. While data is encrypted between the app server and the notification service, and between the notification service and the handset, there is no requirement for it to be encrypted end-to-end. Therefore, app data can often be read by the push notification server. In addition, regardless of support for end-to-end encryption between an app server and a handset, metadata on which handsets use which apps, as well as the location of the user (e.g. via the handset's IP address), are revealed to the notification service.

In this paper we explore the design space of more privacy-friendly designs for push notification services. We consider three broad options: use push notification services as deployed today; connect to a single push notification service via Tor; or run a separate push notification service per app and connect to each of these via Tor. In the latter two cases, connections via Tor could be made outbound from the phone to the service or to a Tor hidden service running on a smartphone. We discuss details of these designs, and the trade-offs they represent, in Section III after we review the background on Tor and its support for hidden services in Section II.

A key requirement for mobile devices is careful management of battery energy and cellular data usage. We therefore measure the data usage costs of using Tor and running a Tor hidden service on an Android handset. We do this by breaking down the costs of running a Tor hidden service into components that allow us to produce a model of data usage as a function of the connectivity profile of a handset. By using connectivity data, such as the availability of WiFi and cellular data from 2 014 handsets in the Device Analyzer [1]

DOI 10.1109/EuroSPW.2017.55

dataset, we estimate the cellular data usage of running a Tor hidden service on an Android device. We find that running a Tor hidden service today costs the median Device Analyzer user 198 MiB per month in cellular data usage, and the top 10% of devices in excess of 362 MiB (and at € 0.20 per MiB, € 72.40 per month). Using EnergyBox [2] we estimate cellular data usage costs 9.6% of battery charge for a Nexus One device with a daily charge cycle.

The cellular data costs of running a Tor hidden service are significant. We therefore explore four strategies to reduce costs, making push notification services over Tor more attractive. By combining these strategies, we are able to reduce the costs of running a Tor hidden service for the median Device Analyzer device to 61 MiB per month.

II. Background

To understand the costs and benefits of using Tor to improve the privacy of push notification services for mobile devices, we start with a brief summary of Tor. The Tor network is composed of *clients*, which generate and consume traffic, including smartphones, laptops, or desktops; and servers, called *relays*, which forward traffic to other relays and make connections to the public Internet on behalf of clients. To use the Tor network, clients download the latest *network status document* approximately every 90 minutes, which lists information about around 7 000 Tor relays currently available worldwide. The network status document is managed by a small number of more trusted servers, called *directory authorities*, who vote on a consensus of its contents once an hour. The directory authorities publish additional, relatively static, information on relays in *relay descriptors* every 18 hours. After downloading the network status document, clients download any relay descriptors mentioned in that document that the client does not already have. The client also downloads certificates of authorities where it does not already have a current one.

Clients use relays to build *circuits* through a sequence of (typically three) relays. Such circuits support an overlay network between the client and the final public Internet service required. The client applies layers of encryption in such a way that none of the relays, nor the final Internet service, is able to determine which devices on the Tor network are connecting to which Internet services. Because circuit construction takes time, clients proactively build circuits in anticipation of any requirement for data connectivity; circuits can also support multiple concurrent TCP streams. Tor clients periodically send keep-alive messages on idle open connections to prevent the connection from expiring at any intermediate routers. The default interval between keep-alive messages is currently 5 minutes. To improve the privacy properties of Tor, circuits are (at least partially) rebuilt every 10 minutes.

A. Tor hidden services

In addition to supporting clients connecting to services such as websites on the public Internet, Tor also allows Tor clients to publish *hidden* services. A hidden service is identified by an *onion address*, which represents the first 16 characters of a base32-encoded SHA1 hash of a public key generated by the client. Onion addresses are long-lived identifiers, distributed through some out-of-band mechanism between parties who wish to communicate. For example, Facebook offers a Tor hidden service at `https://facebookcorewwwi.onion/`.[1] An onion address allows Tor clients to establish circuits with the hidden service using the Tor *rendezvous* protocol, and therefore transfer data to and from the service over the Tor network. The design of Tor hidden services prevents any single relay from learning the IP address associated with an onion address, therefore providing anonymity to both a hidden service provider and its clients.

The rendezvous protocol is described below, where we assume that Bob wants to run a hidden service and Alice wants to connect to it. A reasonably detailed understanding of these steps is required in order to understand the network and energy costs presented in later sections of the paper.

Bob creates a hidden service:

1) Bob asks his Tor client to create a new hidden service. This generates a public-private key pair for the service. The *public key* of the service identifies the service and is used to generate an onion address.
2) Bob shares his onion address with Alice via an out-of-band mechanism.

Bob runs a hidden service:

3) Bob's Tor client chooses a small number of (typically three) relays as *introduction points*. Bob then establishes a circuit to each introduction point and sends a single-use public key, or *service key*, and signs a message to prove he is the owner of this public key.[2] Bob's Tor client must keep the circuits to the introduction points open while the service is running to receive connection requests from new clients.
4) Bob's Tor client generates a *service descriptor* containing the public key, the service key, and the introduction points. The service descriptor is uploaded to a few (currently six) hidden service directories, chosen based on the descriptor ID, which is a hash of the service's public key, the current date and time, and other deterministic data. Bob's Tor client publishes a new descriptor once an hour, or whenever its content changes.

[1] Note: Facebook have spent considerable computational resource to final a public key whose base32-encoded SHA1 is memorable.

[2] Earlier versions used the public key of the hidden service instead of a single-use service key, but this allowed the introduction point to monitor Bob's activity.

Alice connects to Bob's hidden service:

5) Alice's Tor client determines the set of hidden service directories responsible for Bob's key using his onion address and the current time, and retrieves Bob's service descriptor from one of them.
6) Alice's Tor client establishes a *rendezvous* point. It does so by randomly choosing a Tor relay, building a circuit to it, and asking it to act as a rendezvous point, specifying a randomly chosen 20-byte *rendezvous cookie*.
7) Alice's Tor client connects to one of Bob's introduction points and requests an introduction to Bob by providing a hash of Bob's service key. Alice also sends a rendezvous request, including the address of the rendezvous point, the rendezvous cookie, and the first part of a Diffie-Hellman key exchange, all encrypted under Bob's temporary service key.
8) The introduction point forwards the rendezvous request to Bob. Bob checks the request is valid and not a replay.
9) Bob's Tor client creates a new circuit to the rendezvous point chosen by Alice and asks the rendezvous relay to complete a circuit to Alice. Bob's request contains the rendezvous cookie, the second part of the Diffie-Hellman exchange, and a handshake digest. The rendezvous point forwards the latter two to Alice's Tor client. Alice's Tor client checks that the handshake is valid, and both sides derive a new set of keys. A new circuit is now established between Bob and Alice.
10) Alice can now establish one or more TCP connections over her circuit with Bob.

III. Push notifications over Tor

We now consider three overall designs: push notification services as deployed today; connection to a single push notification service via Tor; running a separate push notification service per app and connect to each of these via Tor.

Connecting to a single push notification service may be more energy efficient than using one push notification service per app since separate messages from multiple app servers (likely destined for a variety of apps on the same handset) can be coalesced into a batch for delivery in a single Tor circuit. The downside is that the push notification service learns the app servers (and therefore apps) communicating with a single handset, although it does not necessarily know the identity or location of the handset if such communication is sent over Tor.

Running a hidden service on a smartphone does not, at first glance, appear to provide much benefit over the use of an outbound Tor connection to a push notification service. Importantly, however, hidden services allow app developers to avoid using a push notification service at all if the aim of the app is to share data between client devices.

Mobile devices typically sit behind a NAT or firewall. Thus, direct phone-to-phone communication is often difficult or impossible. If every device operates a Tor hidden service, direct communication between two smartphones is now possible, as an onion address is globally unique and accessible. The downside to this approach is that both the sending and receiving smartphone need to be online simultaneously for data to flow. This requires careful scheduling of smartphones to wake from low-power states and both devices to have network connectivity at the same time. We note that an energy- and data-efficient solution is likely a prerequisite for mobile apps that use device-to-device communication (e.g. messaging apps). We therefore focus on data and energy issues of Tor hidden services. We leave the issue of scheduling communication between devices for future work, although such issues have been addressed before. For example, the PEN network supported direct peer-to-peer communication, with a scheduling algorithm that was more efficient than the more traditional (centralized) master-slave scheme [3, p. 21].

Both connecting to push notification services via Tor, and the use of Tor hidden services, inherit the anonymity properties of Tor, which is resistant to local adversaries who are able to control any local network. This means that a local adversary does not learn the endpoints of any connections. In addition, the app server may also be located behind a hidden service, providing anonymity for the app server too.

Regardless of whether we use a single push notification server, a push notification service per app, or phone-to-phone communication, our primary concern is that using Tor, and possibly running a Tor hidden service, may be significantly less energy-efficient, or may result in substantially more data usage, than traditional push notification services. Quantifying and improving the cost of Tor is a requirement in all three use-cases and is thus the focus of the remainder of this paper.

We note that the use of Tor to support push notifications may increase latency for message delivery, but we do not believe the typical latency times found on Tor will lead to large problems for push notifications. We therefore leave this analysis as an area of future work.

IV. Experimental Method

We present a series of experiments to measure data usage requirements and to estimate the energy consumption of using Tor and operating a Tor hidden service on smartphones. Our testbed consists of two Nexus 5X smartphones running Cyanogenmod (Android 6.0.1). To support the creation and operation of Tor hidden services, we developed a simple custom app that uses Tor project's Orbot Android app (version 15.1.2) to run a hidden service. Our app accepts connections to the hidden service and logs any data sent to it, allowing us to explore data transmission at various rates between the smartphone and another computer. To avoid problems with the phone going into deep sleep, we configured the phone to always stay awake. To provide a comparison with Google's

Cloud Messaging (GCM) service, we installed and enabled the Google Play Services Framework when necessary.

We obtained full packet traces of all traffic on a Linux workstation by connecting the smartphones to a NETGEAR WiFi access point with an Ethernet uplink connected to a workstation. The workstation was configured to route data onto the wider Internet, allowing connections to and from the Tor network and GCM. The experiments where conducted between December 2016 and February 2017.

A. Measuring Tor traffic

To estimate the cost of using Tor for push notifications, we wanted to construct an empirical model of Tor traffic. Such a model is important for accurately estimating the data and energy costs an app might generate using any of the Tor-based push notification systems we discussed in Section III.

As discussed in Section II, the Tor client takes part in many different network activities which we break down into nine categories in order to build an empirical model: regular downloads of the network status; relay (micro) descriptor data; creating circuits to introduction points; regular uploads of hidden service descriptors; sending keep-alive messages along established connections to Tor relays; downloading authority certificates; measuring circuit timeouts; establishing and closing connections to a (first hop) Tor relay; and creating circuits, responding to connection requests, and data communication associated with a hidden service.

In this section we describe how we quantify the amount of network traffic in each above categories. We use this analysis in Section VI to derive an empirical model of Tor data usage and assess the real-world impact of using Tor with handsets in the Device Analyzer project.

Tor traffic is encrypted, and thus it is not straightforward to obtain a breakdown of traffic by category. We therefore instrumented the Tor source code to identify and log the purpose (thus category) of all network data sent or received by the Tor client. We used the log to associate this category with each packet in the network trace captured by the workstation.

Tor clients and routers communicate with one another via TLS connections with ephemeral keys. Traffic on these connections consist of 514-byte *cells*, which contain a header and a payload. Cells are either control cells, used to create, extend, or destroy a *circuit*, or are payload cells, containing encrypted data travelling over an existing circuit. The circuits themselves are used to support connectivity for client applcations (e.g. allowing an app on the phone to make a TCP connection to a push notification service) and maintain connectivity to the Tor network (e.g. downloading network status; uploading a hidden service descriptor; sending a keep-alive message; and so on).

Our instrumented Tor client generally allows us to determine the purpose of each cell sent or received, but associating this with the network trace captured by the workstation is difficult because: multiple cells may be carried inside a single IP packet; a single cell may be split across an IP packet; and TLS handshake messages, TLS headers, TCP headers and TCP re-transmissions introduce additional overhead that should be associated with the underlying category of use.

Accounting for the TCP header size and re-transmissions is relatively easy as these are visible in the packet trace. To account for TLS headers and overheads, we record the number of bytes read and written to the TLS stream and to the underlying TCP socket. We match the byte counts written to the TCP socket with the bytes sent in the network trace to determine which cells (or parts of cells) are contained within a specific network trace. The overheads resulting from TLS and TCP are assigned proportionally to the cells contained within the relevant packets.

Determining the purpose of each cell is generally straightforward since the cell header associates the cell with a specific circuit, and additional instrumentation allows us to record the current purposes of a circuit or of the stream associated with the cell. One complication is that the assignment of a purpose to a cell cannot be made directly after data is read from the underlying TLS connection, since only part of a cell may be returned. Additional bookkeeping is thus needed so that the purpose can be determined after complete cells have been received and parsed. Another difficulty is that many TCP streams can be multiplexed down a single circuit. For circuits that were used for more than one purpose, there can exist some traffic that cannot be assigned to a particular TCP stream (e.g. creating a new circuit); if the purpose cannot be uniquely inferred, the traffic cost is shared equally between all the purposes associated with the circuit.

Consequently, there are two approximations in our analysis that are small and therefore do not have a material impact on our analysis. First, since Tor preemptively builds circuits, some of these circuits may not have been used; we find unused circuits were responsible for only 0.1% of the total traffic. Second, when cells cannot be associated with a TCP stream, and their purpose cannot be inferred, we assign their cost equally to all purposes associated with a given circuit; this only affected 0.2% of the total traffic. Section V-A offers more details.

V. Results

We now report on four experiments. First we measure the cost of maintaining a Tor hidden service for a fixed IP address and stable Internet connection. Second, we measure the additional cost of changing our IP address, a regular occurrence for a smartphone as it moves between cellular data and WiFi networks. Third, we explore the overhead of data transmission across the Tor network. These results allow us to produce a model of the cost of running a Tor hidden service, something we build on in Section VI. Finally, for comparison, we measure the overheads of using GCM.

A. Hidden service maintenance

We measured the network traffic induced by maintaining a Tor hidden service over a 48-hour period using our testbed. We recorded 32.5 MiB of Tor traffic, including IP headers across 46,790 packets, or an average of 693 KiB (975 packets) per hour. The large majority of the traffic volume in bytes was caused by network status consensus downloads (79.9%), with another 11.7% caused by hidden service descriptor uploads. Downloading relay descriptors caused 4.3% of the traffic, keep-alive messages 2.5%, and introduction circuits 0.2%. Establishing and closing connections to entry (first hop) relays was responsible for 0.8% of the traffic. 0.2% was used to measure circuit timeouts, another 0.2% to fetch authority certificates, and the remaining 0.1% was used to manage circuits that remained unused. Table I provides further detail.

At the time of writing, directory authorities vote on a new network status consensus every hour, which is valid for three hours. Clients download a new consensus at a randomly chosen time between 105 and 170.6 minutes after their current consensus becomes valid. We observed a total of 38 consensus downloads, with an average size of 699 ± 9 KiB. In addition, we saw one case where the directory server returned a "304 Not modified" status. In this case, the client retried the download after one minute, resulting in the same status code. When the client tried again at a different server 10 minutes later, it received a full consensus document again. This caused an additional 8 KiB of traffic. We also observed 336 hidden service descriptor uploads. Keep-alive messages are padded to the size of a cell, with the total size of keep-alive IP packets as 595 bytes, which is answered by an ACK packet of 52 bytes. Both sides of the connection send a keep-alive packet, resulting in 4 packets and 1 294 bytes exchanged per idle connection every 5 minutes.

Type of traffic	KiB/h	KiB%	Pkts/h	Pkts%
Network status download	554	79.9%	694	71.2%
Relay descriptors	30	4.3%	47	4.9%
HS descriptor	82	11.7%	144	14.8%
Keep-alive	17	2.5%	54	5.6%
Introduction circuits	1	0.2%	3	0.3%
First-hop connections	6	0.8%	24	2.5%
Measure circuit timeout	2	0.2%	3	0.3%
Authority certificate	2	0.2%	3	0.3%
Unused circuits	1	0.1%	1	0.1%
Total	693	100%	975	100%

Table I
AVERAGE NETWORK TRAFFIC GENERATED WHEN MAINTAINING A TOR HIDDEN SERVICE.

B. Network connectivity changes

Smartphones regularly change their network connectivity as they move between WiFi access points and connections via cellular data services. Whenever such device connectivity changes, connections to the Tor network must be re-established because the source IP address used to support the TCP connections underlying Tor circuits changes.

To estimate the total additional network traffic caused by network connectivity changes, we used the same setup as in the maintenance experiment in Section V-A, but forced a disconnect of the WiFi connection every 20 minutes, and a reconnect 5 seconds later. When Orbot detects that the network is down, Tor shuts down all connections and starts rebuilding connections when connectivity is back.

We then measured the amount of traffic generated over 48 hours and classified it as in Section V-A. Our experiments showed that network status document and relay descriptor downloads were not affected by connectivity changes. We therefore exclude traffic classified as one of these categories. The current implementation of Orbot chooses new introduction points after each reconnect, and re-uploads the hidden service descriptors. Based on Section V-A, which describes the traffic required for a set of hidden service descriptor uploads, we also exclude traffic related to them to get an estimate of the remaining traffic caused by a connectivity change. Ignoring traffic related to these three activities, we calculated the difference in total traffic compared to the idle connection (Section V-A). Excluding these, we measured 5 628 KiB of traffic, compared to 1 362 KiB for the idle connection. During the 48 hour period, the WiFi reconnected 143 times. We therefore estimate an average additional traffic per reconnect of 29.8 KiB, primarily for re-establishing connections, introduction circuits, and other circuits. Adding the approximately 70 KiB it takes to upload hidden service descriptors, a reconnect generates roughly 100 KiB of traffic.

C. Data transmission

We measured the overhead of transmitting data over the Tor network. To do so, we sent messages of three different sizes (1 B, 512 B and 1 KiB) at three different intervals (1 min, 8 min, 12 min) to the smartphone. We chose 8 and 12 minute intervals to explore the effect of circuit rebuilds, which currently occur every 10 minutes (Section II). For each message, we established a fresh TCP connection to the hidden service and sent a stream of bytes of the given length before closing the connection. For each combination, we sent messages for 4 hours. Table II shows how much traffic was generated on average by a single message for different message sizes and rates. We estimated this amount by counting all traffic not labeled as network status download, relay descriptor download, hidden service descriptor upload, certificate authority download, or measuring circuit timeout over the 4 hour-period, subtracting the expected amount of traffic for the same categories for simply maintaining the hidden service as measured in Section V-A (429 bytes/1.4 packets per minute), and dividing by the number of messages sent. Note that we count keep-alive traffic, as receiving messages may reduce or increase the need for keep-alive messages.

Interval	1 B	512 B	1 KiB
1 min	2.7(7.6)	3.2(7.8)	3.8(9.2)
8 min	5.7(15.7)	6.1(16.4)	7.2(19.6)
12 min	9.1(25.1)	9.9(26.7)	9.7(25.4)

Table II
THE AVERAGE ADDITIONAL NETWORK TRAFFIC IN KIB (NUMBER OF PACKETS IN BRACKETS) GENERATED PER MESSAGE OVER TOR FOR DIFFERENT MESSAGE SIZES AND DIFFERENT SENDING RATES.

D. Comparison with GCM

For comparison with Tor, we used our testbed to measure the costs of maintenance, connectivity changes, and message overhead of using GCM. To determine the traffic relevant to GCM, we filtered TCP traffic from the smartphone whose destination was mtalk.google.com, ports 5228–5230.

Push notifications over GCM requires Google Play Services (PS) running on the handset. PS initiates and maintains a single open TCP connection to a GCM server to receive push notifications. To keep the connection alive, PS periodically sends keep-alive messages to a GCM server. The active keep-alive intervals can be determined by typing the code `*#*#426#*#*` in the Phone app. Using this technique, we experimentally confirmed that, for mobile data connections, PS currently uses a 28-minute interval. On WiFi, PS uses a proprietary adaptive algorithm to determine an interval of between 110 seconds and 29 minutes; in our case the interval was typically set to 19, 24, or 29 minutes.

There are no entries in the smartphone system log concerning keep-alive messages. Thus, to quantify data usage and packet count for keep-alive messages, we looked at the packet trace from the smartphone deployed with our testbed with PS installed and enabled. To ensure that PS connected to GCM and waited for push notifications, we wrote and launched a simple app that waits for incoming GCM messages. We observed a periodic burst of three or four packets with a total length between 224 and 278 bytes, which matched the WiFi heartbeat interval. From the 246 bursts we observed, the average total size was 238 ± 22 bytes (not counting duplicate packets). Alongside this periodic burst, we sometimes observed up to four additional packets containing duplicate TCP packets (up to 528 bytes in total). The contents of the packets were encrypted so we could not determine further details of the keep-alive message or the purpose of the retransmission. The average total size including duplicate packets was 258 ± 57 bytes.

We repeated the experiment described in Section V-B for GCM on a Nexus 5X handset. We again forced the phone to reconnect to WiFi every 20 minutes. We ran the experiment for 48 hours. We measured the amount of traffic within a minute after each reconnect and observed a burst of traffic, with an average size of 2.9 ± 1.5 KiB (16.8 ± 1.6 packets) in 141 out of 143 cases when the phone reconnected to WiFi. In two cases, we observed no additional traffic directly after a reconnect. We assume that the fact that we did not change the IP address might have resulted in PS not reconnecting in these cases.

To measure the traffic overhead when sending messages to the smartphone, we sent similar messages to our GCM-enabled app as we did over Tor in Section V-C. We used the same message sizes and intervals (1 min, 8 min, 12 min; 1 B, 512 B and 1 KiB) and we measured each combination for 2 hours. The average traffic per message did not significantly differ for different intervals. Per 1-byte message we observed on average 0.3 KiB, per 512-byte message 0.8 KiB, and per 1024-byte message 1.3 KiB of traffic.

VI. TOR HIDDEN SERVICE MODEL

In this section, we use the results of Section V to derive a model for the data usage of a hidden service on a smartphone. We use this model to evaluate the data usage and energy costs of using Tor to support a push notification service on real devices in Section VI-A. The model is based on the results from our measurements and therefore on the current state of the Tor network. Future work could take into account the changing nature of the Tor network and create a model that depends on parameters like the number of Tor relays that notably impact the amount of network traffic.

To estimate the total network traffic required to maintain the hidden service on a phone, we require knowledge of the connectivity profile of a device: the periods the device was connected to the Internet via WiFi or a cellular network, when the IP address of the handset changes, and when no network connectivity is available. Network traffic is generated by periodic network status and relay descriptor downloads, hidden service descriptor uploads, and the creation and maintenance of Tor circuits to introduction points. We look at each of these in turn.

The network status document is downloaded at regular intervals. Building on our analysis in Section V-A, we assume that the Tor client starts a network status download when either: a disconnected device connects to the Internet and has no valid network status document; or time t (chosen uniformly at random from the interval $[105, 170.6]$ minutes) has passed since their current download became valid. We assume that a new network status document becomes valid on the hour, every hour (in UTC) and the client always downloads the most recent valid document. We assume that each consensus download produces 716 419 bytes of traffic, the average measured in Section V-A.

We assume that the client downloads a set of relay descriptors immediately after it downloads a network status document. We assume that this requires $30\,356 \cdot h$ bytes of traffic, where h is the number of hours that have passed since the last network status download. This is based on the average descriptor download traffic we observed per hour, and should give a good approximation, in particular because

the large majority of descriptor downloads happens shortly after a network status download.

We assume that each time the phone changes the way it connects to the Internet, it needs to rebuild its Tor connections and circuits (including the ones to the introduction points), which costs 30 548 bytes of traffic, the average measured in Section V-B.

We assume that the client uploads its hidden service descriptor immediately after it has (re-)established its connections to the introduction points, or 60 minutes after the last upload. We assume that each set of uploads (to six directories) incurs 71 504 bytes of traffic, the average measured in Section V-A.

Finally, we assume that for every 5 minutes the device is connected to the Internet, 1 474 bytes of keep-alive traffic is generated, the average measured in Section V-A. We do not include periodic changes of the introduction points, as these have a small impact on total traffic (0.2% during the experiment described in Section V-A) and in our analysis, the Tor client changes introduction points once a day. Similarly, for simplicity, we do not include other traffic in our model since the remaining traffic was only about 1% of the total traffic during our measurement.

A. Evaluation

To evaluate the energy and data usage costs of using Tor to support push notifications, we use our model from Section VI together with connectivity profile data of smartphones from the Device Analyzer project [1].

Device Analyzer is an Android app, available on the Google Play Store since May 2011 and installed on over 30 000 handsets. It gathers information on a wide variety of system statistics, including: app usage; metadata on calls placed and received; metadata on text messages sent and received; Bluetooth devices seen and connected to; WiFi access points seen and connected to; cell network coverage for calls and data; and battery and power usage. Data collected by the app is processed on the handset to obscure direct personal identifiers (e.g. phone numbers) before uploading data to a server at the University of Cambridge.

We analyzed traces from the 30 444 devices in the Device Analyzer dataset. We excluded all devices with less than 30 days' worth of data. We further excluded all devices where Device Analyzer data collection has been interrupted at any point, had large jumps in their device clock, or where the device clock was obviously wrong or broken. For each device trace from the remaining 2 014 devices, we estimate the volume of cellular data required to maintain a Tor hidden service. We do this by assuming that cellular data is used when a cellular connection is available and a WiFi connection is not. Since the Tor client uses timing randomisation when downloading the network status document, we simulate the connectivity pattern of the device 40 times and take the average amount of traffic.

The baseline box plot shown in Figure 1 shows our estimate of the cost of running a Tor hidden service for 30 days on the 2 014 devices from the Device Analyzer dataset. An equivalent numeric summary is shown in Table III. Cellular data usage is high, with a median cost across all devices of 198 MiB. For 10% of the devices we estimate a cellular data usage of 362 MiB or more. These are high data rates, and in most countries will require a significant data plan. For exposition purposes, assuming networks charge € 0.20 per MiB, the maximum roaming charge mobile operators within the EU were allowed to charge after March 2014, this represents € 72.40 per month; a substantial price to pay for better privacy. By way of comparison, GCM maintenance, without any IP address changes, costs on average 258 bytes every time it needs to send a heartbeat (see Section V-D), or 0.44 MiB over 30 days for heartbeat interval of 24 minutes. Even factoring in multiple network changes per day, at 2.9 KiB per change, total costs are still likely only a couple of MiB per month.

We use EnergyBox [2] to estimate the energy costs of maintaining a Tor hidden service on a smartphone. EnergyBox takes a packet trace, a smartphone model, and a connectivity profile (WiFi or cellular). To provide a reasonable lower-bound estimate for the energy costs, we reused the 48-hour packet trace collected for our experiment in Section V-A and assumed this trace was transferred over the cellular data network (3G) with a Nexus One device on the TeliaSonera network, the only device and network operator the EnergyBox authors provide an energy model for. The Nexus One was released in 2010, and we expect newer devices' batteries to last longer. Over the 48-hour period, the estimated total energy costs were 5 346 J; or 2 673 J = 0.743 Wh per day. The Nexus One device has a battery capacity of 5.18 Wh, so, assuming a battery profile where the device is charging for 8 hours over night and on battery for 16 hours, this represents 9.6% of total battery capacity – a significant amount. Note that this value only takes into account the energy required for network communication, and additional power is required to make a hidden service work, e.g., to keep the device awake when needed.

B. Reducing Tor data usage

The above numbers demonstrate that running a Tor hidden service on a smartphone generates several hundreds of megabytes of cellular data traffic per month on a typical device, an unacceptable volume for all but those with a generous data package. As the EnergyBox paper [2] demonstrates, there is a strong correlation between data volume and energy usage too. Therefore, we evaluate four strategies to reduce the amount of data Tor requires, thereby reducing energy usage. Note that these strategies may potentially impact user anonymity. We leave evaluating this for future work.

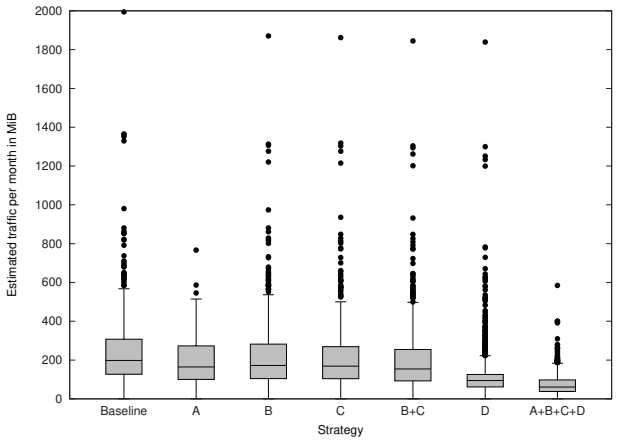

Figure 1. Estimated traffic over the cellular network over 30 days for 2 014 devices in the Device Analyzer dataset. The leftmost box shows a data usage estimate for Orbot's current behaviour; the remaining boxes estimate data usage with various data reduction strategies developed in Section VI-B.

	Base	A	B	C	BC	D	ABCD
Median	198	165	172	169	154	95	61
Mean	367	305	333	322	304	188	119
Std.dev.	141	108	138	132	132	101	45
90^{th} perc.	362	341	353	329	327	172	120
99^{th} perc.	649	424	628	612	607	517	202

Table III
ESTIMATED TRAFFIC IN MIB OVER THE CELLULAR NETWORK OVER 30 DAYS FOR 2 014 DEVICES IN THE DEVICE ANALYZER DATASET.

Strategy A: Reconnect to the same introduction points: In the version of Tor client we used in our experiments, Tor chose a new set of introduction points whenever a device changed its IP address. This is supposed to be fixed [4]. However, we found that changes in network connections continued to require new introduction points. At the time of writing, Tor developers were working on a fix [5].

Changing introduction points requires the Tor client to generate a new hidden service descriptor and upload it. This causes additional traffic, and also affects anyone wanting to connect to the hidden service: other clients may have cached the hidden service descriptor and therefore connectivity is broken. A fix is under development [6].

Strategy A in Figure 1 estimates the data usage costs for a Tor client that reconnects to the same introduction points when the IP address of the smartphone changes, and therefore does not need to re-upload the hidden service descriptor.

Strategy B: Proactively fetch network status on WiFi: If we assume free data usage over WiFi, a straightforward strategy to reduce cellular data costs is to proactively download the most recent network status document as soon as it is available. This has the downside of causing additional traffic at the directory mirrors and will also increase energy costs. Strategy B in Figure 1 explores this option.

Strategy C: Defer fetching network status on cellular: Since mobile devices regularly move between WiFi and cellular data, it may make sense to delay downloading the network status document until just before expiry in the hope that WiFi connectivity appears before a download over the cellular network becomes necessary. Additionally, this may reduce the total number of network status document downloads. This strategy may cause spikes in download requests on directory mirrors in the Tor network if many clients adopt this policy. Nevertheless, strategy C in Figure 1 explores this option.

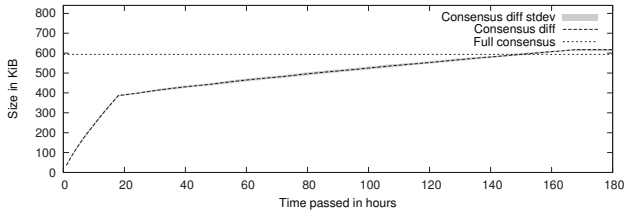

Figure 2. Average size of the compressed output of `diff -d -e` on pairs of all microdescriptor consensus documents from 1 November to 30 November 2016, compared to the compressed size of the full network status documents.

Strategy D: Download network status diffs: The network status document is updated every hour, but only a small part of it changes from hour to hour. Therefore, we consider the potential benefits of downloading the difference in two network status documents. This is not a new idea—a version of this was implemented [7], [8] in 2014—but it is not yet integrated into the main branch of Tor or available in Orbot. To estimate potential savings from downloading differences instead of full documents, we looked at all 720 consecutive network status documents from November 2016 (UTC).

We compared documents pairwise by applying `diff -d -e` and then `gzip -9` to the output to calculate the size of a diff. We calculated how the size of the diff changed as the time period between the pairs of documents increased. Figure 2 shows the average size of the compressed difference between all pairs of documents, grouped by the number of hours between the start of the validity of the two documents. The knee in the curve at 18 hours is due to the fact that Tor relays update their relay descriptors every 18 hours. The data shows that using diffs can drastically reduce download size if time between network status downloads is small. The diff between two consecutive network status documents is a mere 6% of the size of the full network status document; it is still 35% smaller after 18 hours, reaching the size of a full network status download only after 6 days. Thus, consensus diffs can be particularly beneficial for devices that constantly stay connected to the Tor network and frequently download the consensus. To account for the overhead incurred by

downloading the diffs, we add 17.6% to the sizes of all compressed diffs for our model. We calculated this percentage by comparing the average traffic incurred for a consensus download in Section V-A with the average size of compressed consensus documents from November 2016.

Strategy D in Figure 1 computes the cost savings on cellular data if network status diffs followed the averages found in Figure 2.

VII. Related Work

Previous work has highlighted a scalability problem in Tor's design: every client needs up-to-date information on all relays, resulting in a total bandwidth expenditure that grows with the number of clients times the number of relays [9], [10]. Several papers propose more scalable solutions using peer-to-peer architectures [9], [11]–[14]. These approaches are usually based on using distributed hash tables (DHTs) and/or random walks letting clients find random relays on demand without needing the entire list of relays. Mittal et al. [10] proposed an alternative approach to improve Tor's scalability: keep the existing client-server architecture, and letting clients obtain random relays from directory servers or guard relays using private information retrieval techniques. While the focus was to reduce the data usage from network status downloads on the Tor network centrally, such techniques also offer large advantages to mobile devices with limited data usage requirements and energy constraints.

Loesing et al. [15] measured the time taken to complete the steps involved in connecting to a hidden service. Lenhard et al. [16] conducted similar experiments with similar results. They also measured the time a Tor client takes to complete the bootstrapping phase under low-bandwidth conditions. Solberg and Bezem [17] measured various performance characteristics of the Tor network and Tor hidden services, including throughput, access time, connection latency, and reliability, for both the public and a private Tor network.

Wiangsripanawan et al. [18] and Doswell et al. [19], [20] looked at the impact of mobility on the performance of Tor in client mode. They explored the problem of changing network connectivity, and the resulting change in IP address, the need to re-establish Tor circuits, and the loss in connectivity. Wiangsripanawan et al. also considered location privacy. To keep connections alive across changes of IP address, they propose re-establishing a circuit to the same exit node. Doswell et al. estimated the impact on throughput, and proposed client throttling to reduce the amount of "wasted" traffic, and the use of a trusted (private or public) bridge relay to keep circuits open, allowing the client to quickly reconnect to the circuits. Neither of these papers consider the costs of running a hidden service from a mobile device.

Briar [21] is an open source app that uses Tor hidden services to support instant messaging between smartphones without using other cloud infrastructure or push notification servers. They do not quantify the costs of running a hidden service; the results from our paper provide support for the introduction of network status diffs.

VIII. Conclusion

We have shown that the cellular data cost of maintaining a Tor hidden service from a smartphone today is high, with a median cost across all devices of 198 MiB. In the worst case, we see devices with cellular data usage in excess of 600 MiB. Energy costs were also significant: we estimated the network activity would cost at least 9.6% of battery capacity on a Nexus One connected to the Internet via 3G with a daily charge cycle.

We explored four strategies to reduce the cost of maintaining a Tor hidden service on a smartphone: reconnect to the same introduction points when the phone's IP address changes; proactively fetch network status on WiFi; defer fetching network status on cellular connections; and download network status diffs. When these four strategies are combined, the result is a more reasonable total monthly median cost of 61 MiB. However, there remains significant work to do. Our experiments show that Google Cloud Messaging costs in the order of 1 MiB per month, an order of magnitude less than a Tor hidden service with all four of our data reduction strategies deployed. Similarly, transmission of a single 1 KiB message consumes significantly more over Tor (between 3.8 KiB and 9.7 KiB of data; see Table II) as compared with GCM (1.3 KiB; See Section V-D).

The introduction of Doze [22] in Android 6.0 makes some form of privacy-preserving push notification service all the more important. Doze is enabled when the handset is stationary for a period of time, not charging and the screen is off. When Doze is enabled, a handset conserves battery life by suspending apps, including suspending background tasks, network communication, alarms, and wake locks. Consequently GCM is an essential developer tool if an app needs to receive messages from an external source because high-priority messages sent over the network to Google Play Services are not affected by Doze.

Popular apps such as Facebook already allow users to connect via Tor from their smartphone. However, they currently lack support for push notifications for these connections. Our work shows that, with some further work to Tor as outlined in Section VI-B, using Tor for push notifications may be acceptable for such users. Tor hidden services are also likely to be useful to support direct phone-to-phone communication in next-generation apps like Briar [21].

Scaleability is a concern if Tor hidden services become popular for deploying push-notification services. If millions of mobile devices start running a Tor hidden service, the increased bandwidth requirements on the Tor network and the large number of hidden service descriptors that need to be managed by hidden service directories will require more Tor relays to be deployed. This will in turn increase the size of the network status document that needs to be downloaded

to the devices regularly. Future work is therefore required to develop more scalable anonymity networks.

Dataset

Data and source code used to produce the results in this paper are available [23]. Data from Device Analyzer is already available from the Device Analyzer project.

Acknowledgments

Stephan A. Kollmann is supported by Microsoft Research through its PhD Scholarship Programme. Alastair R. Beresford is partly supported by The Boeing Company and EPSRC [grant number EP/M020320/1].

We also thank Nikilesh Balakrishnan, Lucian Carata, and Ripduman Sohan for their assistance with the experimental method; and Martin Kleppmann, Laurent Simon, Daniel RṪhomas, and Diana Vasile for helpful discussion and insight.

References

[1] D. T. Wagner, A. Rice, and A. R. Beresford, "Device Analyzer: Large-scale Mobile Data Collection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 53–56, 2014.

[2] E. J. Vergara, S. Nadjm-Tehrani, and M. Prihodko, "Energybox: Disclosing the wireless transmission energy cost for mobile devices," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 2, pp. 118–135, 2014.

[3] J. Weatherall and A. Jones, "Ubiquitous networks and their applications," *IEEE Wireless Communications*, vol. 9, no. 1, pp. 18–29, 2002.

[4] (2013) Tor Ticket #8239. Hidden services should try harder to reuse their old intro points. https://trac.torproject.org/projects/tor/ticket/8239. Accessed 20 December 2016.

[5] (2016) Tor Ticket #19522. HS intro circuit retry logic fails when network interface is down. https://trac.torproject.org/projects/tor/ticket/19522. Accessed 20 December 2016.

[6] (2015) Tor Ticket #16387. Improve reachability of hidden services on mobile phones. https://trac.torproject.org/projects/tor/ticket/16387. Accessed 20 December 2016.

[7] P. Palfrader. (2008) Provide diffs between consensuses. http://github.com/isislovecruft/torspec/blob/master/proposals/140-consensus-diffs.txt. Accessed 20 December 2016.

[8] D. Martí. (2014) [GSoC] Consensus diffs - Fourth report. https://lists.torproject.org/pipermail/tor-dev/2014-July/007163.html. Accessed 20 December 2016.

[9] J. McLachlan, A. Tran, N. Hopper, and Y. Kim, "Scalable Onion Routing with Torsk," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, 2009, pp. 590–599.

[10] P. Mittal, F. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval," in *Proceedings of the 20th USENIX Security Symposium*. USENIX Association, 2011.

[11] A. Nambiar and M. Wright, "Salsa: A Structured Approach to Large-Scale Anonymity," in *Proceedings of the 13th ACM conference on Computer and Communications Security*. ACM, 2006.

[12] A. Panchenko, A. Rache, and S. Richter, "NISAN: Network Information Service for Anonymization Networks," in *Proceedings of the 16th ACM conference on Computer and Communications Security*. ACM, 2009.

[13] M. Rennhard and B. Plattner, "Practical Anonymity for the Masses with MorphMix," in *Financial Cryptography: 8th International Conference*. Springer, 2004, pp. 233–250.

[14] P. Mittal and N. Borisov, "ShadowWalker: Peer-to-peer Anonymous Communication Using Redundant Structured Topologies," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, 2009, pp. 161–172.

[15] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz, "Performance Measurements and Statistics of Tor Hidden Services," in *2008 International Symposium on Applications and the Internet*. IEEE, 2008, pp. 1–7.

[16] J. Lenhard, K. Loesing, and G. Wirtz, "Performance Measurements of Tor Hidden Services in Low-Bandwidth Access Networks," in *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*. Springer, 2009, pp. 324–341.

[17] P. Solberg and B. Bezem, "Performance of hidden services in Tor," Master's thesis, Norwegian University of Science and Technology, Department of Telematics, 2013.

[18] R. Wiangsripanawan, W. Susilo, and R. Safavi-Naini, "Achieving mobility and anonymity in IP-based networks," in *Proceedings of the 6th International Conference on Cryptology and Network Security*. Springer, 2007, pp. 60–79.

[19] S. Doswell, N. Aslam, D. Kendall, and G. Sexton, "Please Slow Down!: The Impact on Tor Performance from Mobility," in *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2013, pp. 87–92.

[20] S. Doswell, D. Kendall, N. Aslam, and G. Sexton, "A longitudinal approach to measuring the impact of mobility on low-latency anonymity networks," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2015, pp. 108–113.

[21] Briar. https://briarproject.org/. Accessed 20 December 2016.

[22] Optimizing for Doze and App Standby. https://developer.android.com/training/monitoring-device-state/doze-standby.html. Accessed 20 December 2016.

[23] S. A. Kollmann and A. R. Beresford. (2017) Supporting data for "The Cost of Push Notifications for Smartphones using Tor Hidden Services". https://doi.org/10.17863/CAM.7547.