Traffic Backfilling: Subsidizing Lunch for Delay-Tolerant Applications in UMTS Networks

H. Andrés Lagar-Cavilla, Kaustubh Joshi, Alexander Varshavsky,
Jeffrey Bickford[†] and Darwin Parra[*]
AT&T Labs – Research, [†]AT&T Security Research Center, [*]AT&T Mobility

ABSTRACT

Mobile application developers pay little attention to the interactions between applications and the cellular network carrying their traffic. This results in waste of device energy and network signaling resources. We place part of the blame on mobile OSes: they do not expose adequate interfaces through which applications can interact with the network. We propose *traffic backfilling*, a technique in which delay-tolerant traffic is opportunistically transmitted by the OS using resources left over by the naturally occurring bursts caused by interactive traffic. Backfilling presents a simple interface with two classes of traffic, and grants the OS and network large flexibility to maximize the use of network resources and reduce device energy consumption. Using device traces and network data from a major US carrier, we demonstrate a large opportunity for traffic backfilling.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms

Design, Performance

Keywords

3G, UMTS, Traffic Scheduling

1. INTRODUCTION

Wide adoption of mobile devices along with ubiquitous cellular data coverage has resulted in an explosive growth of mobile applications that expect always-accessible wireless networking. This explosion has placed strains on resources that are scarce in the mobile world: handheld battery life and cellular network capacity. On the user side, poor battery life due to unoptimized mobile apps has been blamed for user dissatisfaction and phone returns [7]. On the network side, the growth rate of mobile data is outstripping the rate at which new cellular wireless capacity is being added [6], leading to proposals for optimization of data use through techniques such as WiFi offloading [3].

Little attention, however, has been paid by OS designers to the *efficiency* of the interactions between cellular networks and mobile applications. Because the acquisition of cellular network resources incurs large signaling and latency costs (up to 2 seconds), resources are reserved for several seconds once in use. The extra time at the end of a reservation is usually called "tail", and is a mechanism used to minimize signaling costs when traffic is intermittent yet somewhat closely spaced. Thus, depending on the precise timing of data streams, network usage characteristics such as energy requirements, latency and bandwidth vary widely, even within a single location. Ignoring these dynamics can result in significant waste of scarce network resources and energy - e.g., it has recently been shown that for a popular music streaming mobile application, 3.6% of the traffic consumes 64.1% of the network device energy and produces the vast majority of the network signaling [12].

Proposals to address this issue fall in two camps: a) "tail-optimization" approaches [12, 11] that modify the allocation of network resources to more closely match predicted traffic patterns, or b) traffic scheduling approaches that accumulate traffic into highly-efficient bursts [13, 4]. However, we contend that the opportunities for both tail prediction and traffic shaping are fundamentally limited by one factor: users are unpredictable. They will not prefetch content, or follow perfectly regular patterns to benefit the overall health of the network. Furthermore, delaying traffic from interactive applications can significantly degrade user experience.

An additional problem is that socket-based network APIs provided by current OSes are inadequate for optimizing either network resource allocation or traffic patterns. Sockets hide all the characteristics of the underlying network and provide the abstraction of a pipe over which "instantaneous communication" is available. This abstraction leads to a situation in which applications introduce traffic onto the network at-will, without regard to its impact on device energy, networking signaling, and overall load. Even if a conscientious application were to try and optimize its transmissions, the OS provides no hints on how and when to do so. Further, applications' expectation of instantaneous communication severely limits the OS's or network's ability to schedule traffic. Delays in message delivery can be construed as an indication of failure, and trigger undesirable responses such as quality degradation or failover.

In this paper, we propose *traffic backfilling*, a novel traffic scheduling approach that is exposed by the OS in the form of a separate service. The idea is to allow applications to differentiate network transfers into interactive traffic (e.g., web browsing, instant messaging) and delay-tolerant traffic (e.g., advertisement downloads, synchronization). Today, both kinds of traffic consume the mobile device's energy and network resources with equal priority. In contrast, we propose to exploit naturally occurring network bursts due to interactive traffic, and utilize the network resources reserved by

Mobiheld 2011 Cascais, Portugal

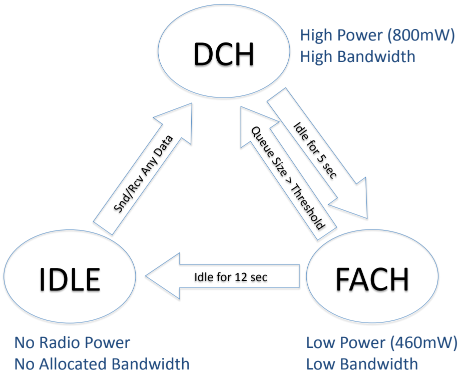

Figure 1: The RRC state machine for "Carrier 1"

those bursts to interject delay-tolerant *backfill traffic* into the gaps between the bursts. The aim is to obtain a "subsidized lunch" in which a large amount of delay-tolerant traffic can utilize resources left over by interactive and unpredictable bursts at a low marginal cost, in terms of device energy and network signaling overhead.

Our primary contributions in this position paper are in quantifying the scope of the backfilling opportunity, sketching the broad outlines of an OS backfill API, and illustrating how existing and new applications may leverage it. Using traces collected from a variety of user devices (phones, laptops with 3G dongles) and operating systems (Mac OS X, Windows, Android), we show that tens of additional MiBs can be transmitted via backfilling even assuming nominally low bitrates (i.e. 256 Kbps). We refine this estimate from a network perspective by analyzing aggregate data from a large urban area on a major carrier's UMTS network comprising 1853 cell sites. We find that aggregate traffic patterns at the per-cell level are highly bursty at small timescales, and are thus amenable to backfilling. For tens of thousand of short-term bursts of network traffic seen by a single cell, with a median length of 16 seconds, the ratio of estimated utilized network resources clusters heavily around only 25% of the peak. With appropriate support, the extra 75% can be reused for backfill traffic.

2. BACKGROUND

We begin by providing an overview of how resource allocation works on UMTS (Universal Mobile Telecommunications System) networks, and its implications for mobile device energy usage and network signaling load. Although we focus on the most widely used 3G technology, the basic principles remain the same for related technologies such as LTE.

UMTS mobile devices establish network connectivity through a UMTS Terrestrial Radio Access Network (UTRAN) consisting of base stations (or Node-Bs) and Radio Network Controllers (RNCs). Each RNC handles a number of base stations and is responsible for controlling network resources. The RNC allocates these resources using per-device RRC (Radio Resource Control) state machines whose transitions are triggered by device traffic exchanges and carrier-determined inactivity timers. For example, Figure 1 shows the state machine for a major US carrier with its parameters summarized in Table 1. This state machine was inferred by the authors of [12] without any carrier help, using measurements made from mobile devices.

FACH → IDLE		12 s
DCH → FACH		5 s
FACH	threshold UL	540
RLC	drain UL	$0.0014t^2 + 1.6t + 20$ ms
Buffer	threshold DL	475
(bytes)	drain DL	$0.1t + 10$ ms
IDLE → DCH signaling		2.0±1.0 s
FACH → DCH signaling		1.5±0.5 s

Table 1: State machine parameters for "Carrier 1"

This state machine has an IDLE state, in which the mobile device consumes no energy on its radio, and no network resources. Upon network use, the mobile device is promoted into a high-bandwidth, high energy consumption DCH state, in which it is allocated a "dedicated" channel by the network. Typical signaling latency for moving to DCH is two seconds, while devices consume between 570 mW to 800 mW in this state [12]. After a few seconds of inactivity, the mobile is demoted to a shared channel, FACH, in the hopes of satisfying marginal network use with a low-bandwidth shared channel, and lower energy usage (450mW in average). Eventually, after a second threshold of inactivity, the mobile is downgraded from FACH back to the IDLE state. If, on the other hand, the mobile networking activity overflows a buffer (in the Radio Link Control layer or RLC), it is promoted again to the DCH state.

Thus, each state of the RRC state machine dictates the device's energy consumption and the network channels it can use. Transitions between low-power states to high-power states are triggered by network activity, whereas transitions between high-power states to low-power states are triggered by inactivity timers. The inactivity timers are necessary to preclude spurious state transitions, which add extra delays and impose non-trivial network signaling overheads. They are tuned by network engineers to provide a healthy dose of hysteresis. There are as many variations on this state machine as there are carriers - with different states, different transitions, and different thresholds for triggering them. But two key elements, staged resource acquisition and hysteresis via inactivity timers, remain across all variations [12].

With HSPA, the latest revision to UMTS, channel capacity can be allocated at a finer granularity within the DCH state. Channel scheduling of devices in the DCH state, both up and downlink, is performed by RNCs with a granularity of 10ms or 2ms [14]. Thus, once in DCH, devices use as many 10ms/2ms scheduling periods as their traffic demands, with little spectrum waste. However, the state machine still remains important because it determines the energy state of the mobile device modem, and the signaling caused by transitions still places non-trivial demands on the network's control plane.

3. TRAFFIC BACKFILLING

We propose backfilling as a new traffic scheduling technique that reuses unutilized gaps between bursts of traffic from interactive applications to send delay tolerant traffic. The OS tracks RRC state machine transitions that result from interactive traffic alone, and transmits backfill traffic during periods of inactivity within the DCH state. Such unused DCH periods exist because of the hysteresis imposed by the RRC state machine inactivity timers. Figure 2 shows unused periods both between gaps in bursty traffic, and during the whole duration of the "DCH tail," before the demotion to the FACH state. By using such gaps profitably, the device can exchange large amounts of additional data with little additional energy usage and signaling load.

To achieve the lowest incremental cost, backfilling should closely track the state machine transitions that would have occurred with interactive traffic only. Backfill traffic must be ignored when computing the state machine trigger conditions (Table 1). Although the state machine is maintained within the RNC, the trigger conditions

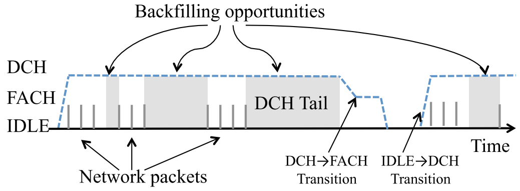

Figure 2: Backfilling reuses unused gaps between interactive traffic bursts

we need can also be controlled, indirectly, entirely from the device itself through the use of fast dormancy [1]. Fast dormancy is a mechanism to allow devices to signal an immediate DCH to the IDLE transition to the RNC. When the OS detects that the RRC state machine would have down-transitioned if backfill traffic were absent, it can initiate fast dormancy to transition to the IDLE state, thus emulating the state machine's behavior in absence of backfill traffic.

Improper use of fast dormancy can substantially increase the frequency of RRC state changes, and overload the network control plane by increasing signaling load [8]. Therefore, network carriers often work with vendors to disable or significantly restrict the use of fast dormancy. Preliminary investigation shows that none of the UMTS USB dongles available for a major US carrier had a fast dormancy API that was exposed to the OS; however, an undocumented AT command was discovered for Infineon chipsets [9]. Unlike tail optimization approaches [12, 11, 4] that advocate shortening of DCH tails, our approach is compatible with such carrier restrictions on fast dormancy since we seek to use it to *preserve* the original state-machine transitions.

Delaying traffic within the mobile device only covers outbound traffic. Handling inbound traffic entails the complexity of requiring the remote sender to pause the network flow when there is no opportunity for backfilling. For short durations, this can be accomplished without cooperation from the sender by using the TCP persist condition [5]. Specifically, the mobile OS can "stall" (resume) a TCP flow by advertising a zero (non-zero) length TCP receiver window. Since the mobile device is unreachable when the flows are stalled, network support for responding to sender probe packets for stalled flows may be required. While all TCP implementations are required to support zero-sized windows [5], firewalls or applications can sometimes terminate long-lived stalled connections to reclaim resources. For such situations or UDP flows, more extensive network staging of inbound flows may be needed.

The resulting delay-tolerant backfill service is an "eventually complete" best-effort service that, at the OS or network's discretion, makes arbitrary progress in reliably transmitting a stream of bytes to the receiver. Progress is not guaranteed - for instance, with the proper network support, the network can ask to delay transmission of backfill traffic in periods of network congestion. Conversely, the OS may batch and efficiently send a large chunk of backfill traffic even in the absence of any interactive traffic. In return for the flexibility it provides in handling network congestion, network providers may choose to provide economic incentives for backfill data either by charging lower per-byte rates, or by counting only a fraction of the transfer towards a user's data cap.

4. API PROPOSAL

One important advantage of backfilling is the simplicity of the API exposed by the OS. At its core, the backfilling API allows applications to discriminate between interactive and delay-tolerant traffic. We outline here a proposal for such API; without loss of generality, we focus on Linux.

The traditional method for establishing network connections is the socket interface, which allows for fine-grained control via the `setsockopt` family of system calls. An initial `setsockopt` can be used by applications to tag sockets as transmitting delay-tolerant traffic. An application can move the connection out of delay-tolerant queues through an explicit flag setting, or as a side-effect of other socket control settings such as disabling Nagle's algorithm.

Unbounded wait is never desirable for delay-tolerant traffic. Most delay-tolerant traffic expects to be transmitted "soon" as opposed to "right now", and perhaps a more accurate term would be "non-urgent" traffic. Two bounds to trigger transmission of the accumulated delay-tolerant traffic are *wait time* and *bytes accumulated*. The former will trigger transmission if the oldest packet has waited for more than N seconds; the latter will transmit once the byte count exceeds a B threshold. Bounds will also cause the same traffic aggregation effect sought by application modification in [12], and the "taps" and "preserves" primitives of Cinder [13]. Bounds can be applied globally, or on a per-socket basis.

So far, our discussion has been application centric. Yet, the OS can easily detect periods of active resource reservation for which there is no pending traffic to backfill. A callback mechanism allows the OS to signal to applications this transient opportunity to push further payloads on a backfilling window. The `select` family of system calls provides a good conduit: it allows applications to (i) decide when to wait, (ii) aggregate multiple events in a single wait point, and (iii) read from the OS via a pipe additional information, such as the size of the current backfilling window.

In the interest of successful upstreaming, particularly for a kernel like Linux, we can stay away from changes to the core kernel socket routines. Instead, we can manage buffering of payloads, aggregation, and completion notification entirely within the bounds of a user-space toolkit. The responsibilities of the kernel are limited to prioritizing packets from interactive sockets, tracking of the RRC state machine, and notifying state changes to user-space.

4.1 Applications

On top of the OS interface, a toolkit can expose multiple user-space primitives to simplify application interaction with the backfilling machinery. Applications can set up staging areas for outbound payloads; they can set up a publish-subscribe mechanism to be alerted of arrival of messages. Or, more generally, they can structure themselves as event-driven loops with callbacks reacting to backfilling activity. While existing applications can benefit from transferring "subsidized" delay-tolerant data, we also believe backfilling will enable applications that users may be reluctant to run otherwise. We outline four applications and the primitives they can leverage:

Participatory Sensing: The plethora of sensors (GPS, accelerometer, camera, etc) on contemporary smart phones has resulted in the birth of participatory sensing applications, in which phones upload sensor readings to data-mining repositories. Such uploads can use a staging area to aggregate data until a backfilling opportunity arises.

Backup and Synchronization: With backfilling, backup can be performed opportunistically and at a possibly lower price. This is a classical example of an event-driven application, triggering backup cycles upon availability of backfilling windows.

Email and RSS: Periodic email and RSS polling have been used frequently in the literature [4, 13] as examples of delay-tolerant

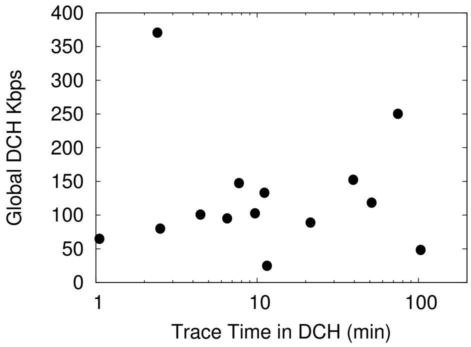

Figure 3: DCH utilization vs. DCH time for 15 traces of varying duration.

traffic that can be optimized for 3G networks, through the use of staging areas (for outbound email) and publish-subscribe pools (for inbound RSS and email).

Cloud offload: Recent proposals [10] have recommended offloading security checking to cloud services, at a high data transmission cost. Backfilling can turn such services into event-driven loops and lower or eliminate transmission costs. The amount of tolerable delay for security responses can be easily tuned through the API.

5. THE BACKFILLING OPPORTUNITY

We build support for backfilling by measuring the size of the opportunity in two phases. We first focus on device-centric measurements, and then later extend the study to incorporate network wide metrics.

5.1 Device-Centric Measurement

We harvested fifteen traces of UMTS networking traffic. We utilized a Windows 7 laptop with an Infineon-based LG Adrenaline data card, a Macbook Air laptop with an Option data card, and an Android-based Samsung Captivate phone with an Infineon built-in modem. All modems are HSPA+ capable. In each case, these devices already had an owner with established usage patterns. We simply turned on tracing, disabled WiFi and Ethernet to ensure the UMTS network was used, and let the user continue life as usual. The durations of the traces were randomly distributed between as little as five minutes, to over an hour (77 minutes), to three full-day traces (around 1420 minutes).

We used libpcap-based utilities (tcpdump, wireshark, etc) to harvest our data traces. Because libpcap presents packets as seen by the OS, HSPA-level MAC headers and retransmissions are not captured. We then fed the traces to an emulated UMTS radio resource state machine, following the parameters of Figure 1 and Table 1 [12]. Assuming the traffic had occurred in the Carrier 1 network (we do not know whether that is the case), we calculated the time spent by the network devices in DCH and FACH modes, as well as the bytes transmitted in each case.

The numbers obtained are consistent with measurements in the literature [12, 11]. DCH tail times hover around 20%. Backfilling opportunities are not limited to DCH tails – any significant gap in network use during a DCH state can be used. For example, one of our traces corresponds to roughly 75 minutes of watching a Hulu video. The UDP traffic from Hulu peers prevents the modem from ever leaving the DCH state. Yet, packet inter-arrival times of two seconds are not uncommon.

In Figure 3 we plot each trace, sorted by the amount of time spent in DCH state, against the global DCH bitrate in Kbps (i.e. bits sent and received in DCH, divided by DCH time). Rarely does a trace exceed 150 Kbps of global DCH bitrate. Most traces accumulate longer than 10 minutes of DCH time, with a trace showing as little as 25 Kbps over 11 minutes. Clearly, the opportunity for backfilling while spending marginal additional energy is present, even for nominal channel capacity as low as 384 Kbps, which was already achievable a decade ago [2]. At that capacity, most of our traces could push over 20 MiBs of backfill traffic.

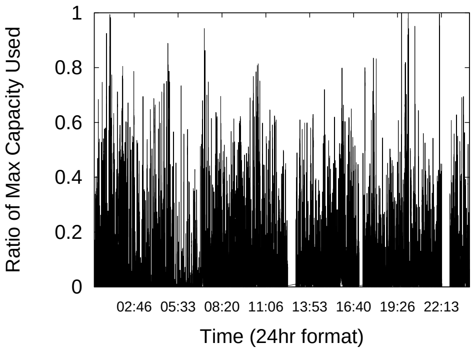

Figure 4: Ratio of data demand per sec in a single cell to the peak demand per sec over 24 hours.

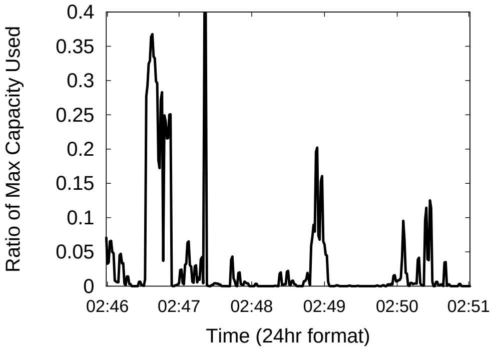

Figure 5: Random five second period in Figure 4.

5.2 Network-Centric Measurements

To complement the view from the device perspective, we now turn to network traces. We show results from a UMTS provider in a major US urban area for May 15th 2011. The data feed we obtained is decomposed by RNC, by Node-B within each RNC, and by mobile within each cell, totaling 1853 Node-Bs and 130653 mobile users. The data for each mobile consists of the number of kilobits sent and received for every second for which there was DCH activity (i.e., the data demand). Due to business reasons, we cannot reveal the urban area, nor the absolute capacity measurements – hence all numbers are expressed as ratios and percentages.

In Figure 4 we chose a particular cell that experienced a high degree of activity. We plot the demand per second over the full day, expressed as a ratio of the peak, which is achieved at about 7:42pm. We note the high short term variance in demand, which reveals a substantial opportunity for backfilling. Figure 5 zooms into a five second period (starting at 10 thousand seconds) to reveal the abundance of short-term valleys of low utilization.

Feeding the data harvested for each individual mobile to a state machine emulator gives a finer-grained picture. For each mobile and each DCH period, we can calculate spare capacity that can be reused for backfilling. The individual peak bitrate actually achieved

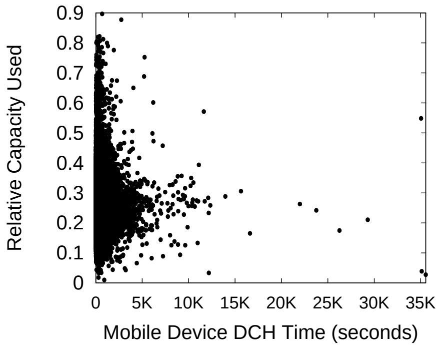

Figure 6: Relative capacity used (per-mobile).

by each mobile device in each DCH period is used as a proxy for the maximum achievable bitrate. This is a conservative estimate, and therefore our estimates of the backfilling opportunity are also conservative. Figure 6 plots the ratio of the used capacity to total capacity for each DCH period in each mobile. The relative capacity utilized is largely independent of the length of the DCH periods, and clusters heavily around 25%. In other words, for most DCH periods, three additional bytes could have been backfilled for each byte transmitted over UMTS, at marginal additional energy cost for the device.

Finally, Figure 7 shows a cumulative distribution function (CDF) of the ratio of the unused DCH capacity represented by DCH tails. The plot shows that over half the opportunities for backfilling lie beyond the DCH tails, with unused capacity available in the "spaces in between" in a transmission stream. The median ratio represented by DCH tails is slightly over 40% of the unused DCH capacity. The CDF shows that for roughly 15% of the DCH periods, the DCH tail represents all unused capacity. For such instances, the period of active DCH use is the minimum quantum recorded by our tracing machinery, thus representing isolated single-shot bursts of networking activity.

6. CONCLUSIONS

In this position paper we have argued for *traffic backfilling* as a means to allow applications to optimize their interactions with wireless cellular networks. With backfilling, delay-tolerant traffic can be transmitted leveraging the unused resources left over by bursts of interactive and urgent foreground application traffic. We have shown through device traces and network data from a major US carrier that there is ample opportunity for traffic backfilling today, at a marginal cost both from a network signaling and device energy standpoint. Our future work focuses on realizing the implications of this position paper, and enabling new mobile applications otherwise thought to be impractical.

7. REFERENCES

[1] 3GPP. Fast Dormancy: a Way Forward. http://bit.ly/itBSjj.

[2] 3GPP. Release 1999. http://bit.ly/lMYpxT.

[3] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting Mobile 3G Using WiFi. In *Proc. Mobisys*, June 2010.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. IMC*, Chicago, IL, Nov. 2009.

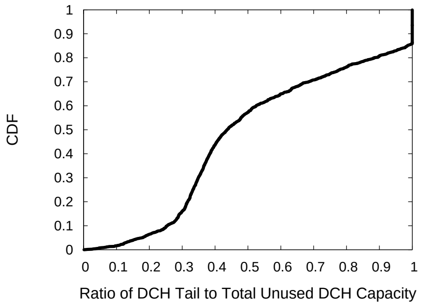

Figure 7: CDF of ratios that DCH tails represent out of the total unused capacity in DCH periods.

[5] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093.

[6] F. C. Commission. Mobile Broadband: The Benefits of Additional Spectrum. http://bit.ly/9ia36u, Oct. 2010.

[7] Electronista. Motorola Chief Pins Android Phone Returns on Poor Apps. http://bit.ly/lMByCF.

[8] FierceWireless.com. What's Really Causing the Capacity Crunch? http://bit.ly/bUN9MX.

[9] Ofono.org. Adding Fast Dormancy Support to Infineon Modem. http://bit.ly/lrthJy.

[10] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile Protection For Smartphones. In *Proc. 26th ACSAC*, 2010.

[11] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. TOP: Tail Optimization Protocol for Cellular Radio Resource Allocation. In *Proc. ICNP*, Tokyo, Japan, Oct. 2010.

[12] 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*, Washington, DC, June 2011.

[13] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich. Energy Management in Mobile Devices with the Cinder Operating System. In *Proc. of Eurosys*, Salzburg, Austria, Apr. 2011.

[14] Wikipedia. Transmission Time Interval. http://bit.ly/kNIyMH.