TOP: Tail Optimization Protocol
For Cellular Radio Resource Allocation

Feng Qian[1] Zhaoguang Wang[1] Alexandre Gerber[2] Z. Morley Mao[1] Subhabrata Sen[2] Oliver Spatscheck[2]

[1]University of Michigan   [2]AT&T Labs - Research

Abstract—In 3G cellular networks, the release of radio resources is controlled by inactivity timers. However, the timeout value itself, also known as the *tail time*, can last up to 15 seconds due to the necessity of trading off resource utilization efficiency for low management overhead and good stability, thus wasting considerable amount of radio resources and battery energy at user handsets. In this paper, we propose Tail Optimization Protocol (TOP), which enables cooperation between the phone and the radio access network to *eliminate* the tail whenever possible. Intuitively, applications can often accurately predict a long idle time. Therefore the phone can notify the cellular network on such an imminent tail, allowing the latter to immediately release radio resources. To realize TOP, we utilize a recent proposal of 3GPP specification called *fast dormancy*, a mechanism for a handset to notify the cellular network for immediate radio resource release. TOP thus requires no change to the cellular infrastructure and only minimal changes to smartphone applications. Our experimental results based on real traces show that with a reasonable prediction accuracy, TOP saves the overall radio energy (up to 17%) and radio resources (up to 14%) by reducing tail times by up to 60%. For applications such as multimedia streaming, TOP can achieve even more significant savings of radio energy (up to 60%) and radio resources (up to 50%).

I. INTRODUCTION

In cellular networks, the release of radio resources is controlled by inactivity timers. However, the timeout value itself, also known as the *tail time*, can last up to 15 seconds [9]. This value is usually empirically chosen to balance the tradeoff among radio resource utilization, user experience, energy consumption, and network processing overheads, based on observed traffic patterns [9]. The tail time is an idle time period corresponding to the inactivity timer value before radio resources are released, leading to waste in radio resources in the cellular network and battery energy of *user equipments* (UEs, *i.e.,* handsets). Based on measurements collected from a large commercial cellular provider, we found that 34% of the occupation time of the high-speed dedicated transmission channels is wasted on the tail, as a result of the bursty nature of the traffic.

We focus on the UMTS (Universal Mobile Telecommunications System) 3G network, which is among the most popular 3G mobile communication technologies. To manage radio resources, UMTS maintains an RRC (radio resource control) state machine for each UE device. A UE can be in one of three *states*, each with different amounts of allocated radio resources, affecting user experience and UE energy consumption. A UE always experiences a tail, its length determined by the inactivity timer, whenever the state is *demoted* from a state with a larger amount of resources to one consuming less

resources. On the other hand, frequent state *promotions* may lead to unacceptably long delays for UEs due to additional processing overheads for the radio access network.

In this paper, we address the problem of mitigating the tail effect in UMTS networks. Existing approaches for tail removal can be classified into three categories.

Tuning inactivity timers. Previous work [14], [23] propose tuning inactivity timers using analytical models by considering radio resource utilization, UE energy consumption, service quality, and processing overheads of the radio access network. However, mitigating the tail effect requires reducing the inactivity timers. Doing so inevitably causes the number of state transitions to increase. Based on our measurements using real traces collected from a large UMTS carrier, we found that aggressively reducing the most critical timer from 5s to 0.5s reduces the tail time by 50%, but increases the state promotion delay by about 300%. This introduces significant processing overhead for the radio access network [4], and increased delay for users.

UE-based approach. The UE alters traffic patterns based on the prior knowledge of the RRC state machine. For delay-tolerant applications such as Email and RSS feeds, data transfers can be delayed and batched to reduce the tail time [6]. However, such an approach is not suitable for more interactive applications such as Web browsing, otherwise users may suffer from delayed processing of their requests.

Cooperation between the UE and the network. The UE applications may be able to predict the end of a data transfer based on the application logic. If an idle time period that lasts at least as long as as the inactivity timer value is predicted, the UE sends a message to notify the network, which then immediately releases allocated resources. This approach can thus completely eliminate the tail if the prediction is accurate, without incurring additional promotion delays. A feature called *Fast Dormancy* has been proposed to be included in 3GPP [2] to help realize this approach. Note that although this is a standard already adopted by several handsets [3], to the best of our knowledge, no smartphone application to date uses fast dormancy, partly due to a lack of OS support.

In this paper, we propose Tail Optimization Protocol (TOP), an application-layer protocol that bridges the gap between the application and the fast dormancy support provided by the network. Some of the key challenges we address include the required changes to the OS, applications, and the implication of multiple concurrent connections using fast dormancy. In particular, TOP addresses three key issues associated with

allowing smartphone applications to benefit from this support.

First, our work is the first to propose a simple interface for different applications to leverage the fast dormancy feature. In our framework, applications define their logical *transfers* and perform predictions of inter-transfer times. The prediction can be easy accomplished for applications having limited or no user interaction (*e.g.,* video streaming), but it is more challenging for user-interactive applications such as Web browsing. The prediction methodology is not our focus in this work. The application invokes a tail removal API provided by TOP that automatically coordinates concurrent traffic of multiple applications, as state transitions are determined by the aggregated traffic of all applications. Our design minimizes applications' implementation overhead for tail removal. Note that our proposed framework is also applicable to the 3G EvDO (Evolution-Data Optimized) and the 4G LTE (Long Term Evolution) cellular networks that also use inactivity timers for releasing radio resources and therefore have the tail effect [7], [19].

Second, by using cellular traces collected from a large UMTS carrier, we are the first to quantify the tail effect for nearly a million user sessions. We found that for the two RRC states, 34.4% and 76.8% of the time is spent on the tail. By using the traces, we also empirically derive critical parameters used by TOP to properly balance the tradeoff between the resource saving and the state transition overhead.

Third, we demonstrate the benefits of TOP using real traces collected from a UMTS carrier and from our Android smartphones. With a reasonable prediction accuracy, TOP saves the overall radio energy (up to 17%) and radio resources (up to 14%) by reducing up to 60% of the tail time. For some applications such as multimedia streaming, TOP can achieve even more significant savings of radio energy (up to 60%) and radio resources (up to 50%).

II. Background

As illustrated in Figure 1, the UMTS network consists of three subsystems: User Equipment (UE), UMTS Terrestrial Radio Access Network (UTRAN), and the Core Network (CN) [13]. UEs are essentially mobile handsets. UTRAN allows connectivity between UEs and CN. It consists of two components: base stations, which are called Node-Bs, and Radio Network Controllers (RNC), each controlling multiple Node-Bs. Most UTRAN's features (packet scheduling, radio resource control, handover control, *etc.*) are implemented at the RNC. The centralized CN can be regarded as the backbone of the cellular network.

In the context of UMTS, the *radio resource* refers to WCDMA codes that are potential bottlenecks of the network. To efficiently utilize the limited radio resources, the UMTS radio resource control (RRC) protocol introduces a state machine associated with each UE. There are typically three RRC states [12].

IDLE is the default state when a UE is turned on. The UE has not established an RRC connection with the RNC, thus no radio resource is allocated and a UE cannot transfer any user data.

CELL_DCH. The RRC connection is established and a UE is usually allocated *dedicated* DCH transport channels in both downlink (DL, RNC→UE) and uplink (UL, UE→RNC). This state allows a UE to fully utilize radio resources for user data transmission. We refer to CELL_DCH as "DCH" thereafter. When a large amount of UEs are in DCH state, radio resources may be exhausted due to the lack of channelization codes. Then some UEs have to use low-speed shared channels although their RRC states are still DCH. A UE can access HSDPA/HSUPA (High Speed Downlink/Uplink Packet Access) mode, if supported by the infrastructure, at DCH state. For HSDPA, the high speed transport channel is not dedicated, but shared by a limited number (*e.g.,* 32) of users [12].

CELL_FACH. The RRC connection is established but there is no dedicated channel allocated to a UE. Instead, the UE can only transmit user data through shared low-speed channels that are typically less than 20kbps. We refer to CELL_FACH as "FACH" from this point on. FACH is designed for applications requiring very low data throughput rate.

RRC states impact a UE's radio energy consumption. A UE at IDLE state consumes almost no energy from its wireless network interface. While within the same state (DCH or FACH), the radio power is fairly stable regardless of the data throughput when the signal strength is stable. Except during transient and error situations, the state machine is synchronized at both the UE and the RNC. Also both the downlink (DL) and the uplink (UL) use the same state machine.

There are two types of RRC state transitions. *State promotions*, including IDLE→FACH, IDLE→DCH, and FACH→DCH transitions, switch from a state with lower radio resource and UE energy utilization to another state consuming more resource and UE energy. *State demotions*, consisting of DCH→FACH, FACH→IDLE, and DCH→IDLE transitions, go in the reverse direction. Depending on the starting state, a state promotion is triggered by either any user data transmission activity, if the UE is at IDLE, or the per-UE queue size, called Radio Link Controller (RLC) buffer size, exceeding a threshold in either direction, if the UE is at FACH.

The state demotions are triggered by two inactivity timers configured by the RNC. We denote the DCH→FACH timer as α, and the FACH→IDLE timer as β. At DCH, the RNC resets the α timer to a fixed threshold T whenever it observes any UL/DL data frame. If there is no user data transmission for T seconds, the α timer times out and the state is demoted to FACH. A similar scheme is used for the β timer.

Promotions involve more work than demotions do. In particular, state promotions incur a long "ramp-up" latency of up to 2 seconds during which tens of control messages are exchanged between a UE and RNC for resource allocation. Excessive state promotions increase processing overheads at the RNC and degrade user experience, especially for short data transfers [4], [18].

Figures 2 and 3 depict state transition diagrams for two large UMTS carriers denoted as Carrier 1 and Carrier 2, whose state machine parameters (under good signal strength conditions) are listed in Table I. They are inferred by our measurement
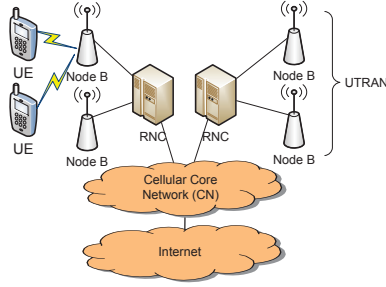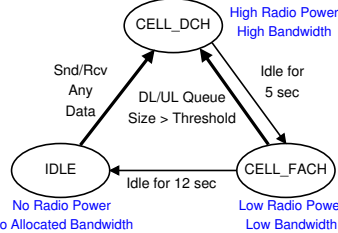
Fig. 1.   The UMTS architecture
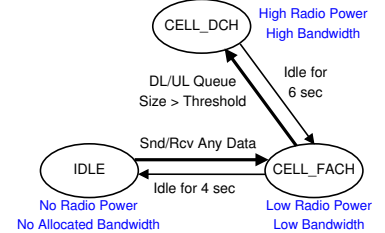

Fig. 2.   RRC state machine for Carrier 1

Fig. 3.   RRC state machine for Carrier 2

TABLE I
STATE MACHINE PARAMETERS FOR TWO MAJOR CARRIERS (FROM [18])

Inactivity timer	Carrier 1	Carrier 2
α: DCH→ FACH	5 sec	6 sec
β: FACH→ IDLE	12 sec	4 sec
Promotion time	**Carrier 1**	**Carrier 2**
IDLE → FACH	N/A	0.6 sec
IDLE → DCH	2 sec	N/A
FACH→ DCH	1.5 sec	1.3 sec
RLC Buffer threshold	**Carrier 1**	**Carrier 2**
FACH→ DCH(UL)	543 ± 25 B	151 ± 14 B
FACH→ DCH(DL)	475 ± 23 B	119 ± 17 B
State radio power	**Carrier 1**	**Carrier 2**
DCH / FACH / IDLE	800/460/0 mW	600/400/0 mW
Promotion radio power	**Carrier 1**	**Carrier 2**
IDLE → FACH	N/A	410 mW
IDLE → DCH	550 mW	N/A
FACH→ DCH	700 mW	480 mW

work [18] and have been thoroughly validated. In this study, we use both in our state machine simulator for cellular traces (§IV) to characterize the tail effect.

III. TOP OVERVIEW

The high-level idea behind our proposed Tail Optimization Protocol (TOP) is straightforward. It involves invoking the fast dormancy support (§V-C) that directly triggers a DCH→IDLE or a FACH→IDLE demotion without experiencing timeout periods, in order to save radio energy and radio resources. However, doing so aggressively may incur unacceptably long delay of state promotions, worsening user experience and increasing processing overheads at the RNC. TOP employs a set of novel techniques to address this key challenge by letting individual applications predict tails and coordinating tail prediction of concurrent applications for invoking fast dormancy. Our design requires no changes at a UE's firmware/hardware given that fast dormancy is widely deployed, and is transparent to the radio access network.

- TOP leverages the knowledge of applications that *predict* the idle period after each data transfer. The definition of a data transfer depends on the application. Fast dormancy is not invoked if the predicted idle period is smaller than a predefined threshold called *tail threshold* to prevent unnecessary state promotions (§VI-A).
- As described in §VI-C, we carefully tune the value of the tail threshold and other parameters used by TOP by empirically measuring traces collected from a large UMTS carrier (§IV), in order to well balance the tradeoff

(§V-B) between the incurred state promotion overhead and resource savings.

- The RRC state transitions are determined by the *aggregated* traffic of all applications running on a UE. TOP introduces a novel coordination algorithm to handle concurrent network activities. TOP also handles tail optimization for legacy applications that are themselves unaware of TOP. We detail the coordination algorithm design in §VI.

IV. THE MEASUREMENT DATA

This section describes the data used in our study. Our dataset is a large TCP header packet trace collected from Carrier 1 on April 13, 2009 in the normal course of operations. The collection point is at the core network (CN) that primarily serves UMTS users but also 2G GPRS users. Our trace contains 265 million TCP packets (169 GB data) continuously captured in 1.3 hours without any sampling in either direction. Due to concerns of the large traffic volume and user privacy issues, we only recorded TCP/IP headers and a 64-bit timestamp for each packet, but no subscriber IDs or phone numbers.

We subsequently extract *sessions* from the trace, each consisting of all packets transferred by the same UE identified through the private client IP address in the trace. Multiple TCP flows from concurrent applications may be mixed in the same session. We use a threshold of 60 sec of idle time to detect session termination. A different threshold value, *e.g.,* 45 or 75 sec, does not qualitatively affect the analysis results.

We will use this dataset in §V-A, §VI-C, and §VII-A. Our common methodology is to replay sessions against a program simulating the RRC state machine with desired settings, and a tail removal algorithm to be studied, *e.g.,* TOP, to obtain statistics about the state machine's behavior. Before that, timestamps of the original trace were first *calibrated* to eliminate promotion delays caused by the existing state machine of Carrier 1. We detail the calibration methodology in our measurement work [18]. Then the calibrated trace, whose promotion delays are zero, can be applied to a different state machine, and new promotion delays are injected separately by the simulator. The calibration procedure also detects sessions (about 17%) that violate the RRC state machine. Such sessions are mostly caused by non-UMTS traffic mixed in the trace. They are not used in our subsequent data analysis.

V. THE TAIL EFFECT

At DCH or FACH, when there is no user data transmission in either direction for at least T seconds, *i.e.,* the inactivity
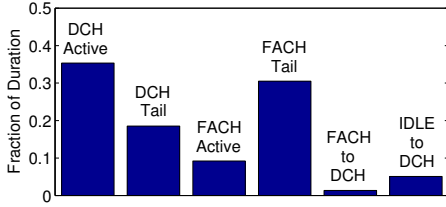
Fig. 4. Breakdown of the duration of all sessions for Carrier 1

timer value, the RRC state will be demoted to save radio resources and UE's energy. However, during the wait time of T seconds, a UE still occupies the transmission channel and WCDMA codes, and its radio power consumption is kept at the corresponding level of the state. We define a *tail* as the idle time period matching the inactivity timer value before a state demotion. We also refer to any non-tail time as *active*.

In typical UMTS networks, each UE is allocated dedicated channels whose radio resources are completely wasted during the tail time. For HSDPA [12], which is a UMTS extension with higher downlink speed described in §II, although the high speed transport channel is shared by a limited number of UEs (*e.g.,* 32), occupying it during the tail time can potentially prevent other UEs from using the high speed channel. Furthermore, tail time wastes a UE's radio energy, which contributes to up to half of a UE's total battery energy consumption based on our measurements using a power meter.

A. Measuring the Tail Time

We quantify the tail time by studying the trace described in §IV. We feed the 0.82 million calibrated sessions into a simulator for Carrier 1's state machine whose state machine transitions and parameters are shown in Figure 2 and Table I. We decompose the total duration of all sessions into six components: DCH active time, DCH tail time, FACH active time, FACH tail time, the FACH→DCH promotion delay, and the IDLE→DCH promotion delay, and then plot the fraction of each component in Figure 4, which clearly shows that considerable amount of time on DCH and FACH is wasted by the tail effect. On DCH, 34.4% of the time belongs to the tail, and on FACH, 76.8% of the time is spent on the tail, which is even longer than the FACH active time, as the β (FACH→IDLE) timer is set to be as long as 12 seconds. For Carrier 2, 35.3% and 72.0% of the DCH and FACH time belong to the tail, respectively.

B. Tradeoff Considerations to Optimize Radio Resources

From the carrier's perspective, the most naïve way to mitigate the tail effect is to reduce the inactivity timer values. However, doing so increases the number of state transitions. As described in §II, completing a state promotion takes up to 2 seconds during which tens of control messages are exchanged between a UE and RNC [12]. Such a delay degrades end user experience and increases the RNC's CPU processing overhead, which is much higher for handling state transitions than for performing data transmission [12].

In order to quantify this key tradeoff, we compute the following four metrics: D^T, D, S, and E. *(i)* The DCH tail time D^T captures radio resources (WCDMA codes) wasted

on tails that can potentially be saved. *(ii)* The total DCH time D consists of DCH tail time (D^T) and DCH active time (non-tail time). It quantifies the overall radio resources consumed by UEs on dedicated DCH channels (we ignore radio resources allocated for shared low-speed FACH channels). *(iii)* The total promotion delay S is the total duration of all promotions. It abstracts the overhead brought by state promotions that worsen user experience and increase processing overheads at the RNC. *(iv)* The radio energy consumption E is the total radio energy consumed in all states and state promotions. The key tradeoffs of radio resource optimization can then be stated as follows. *Increasing (decreasing) inactivity timers causes ΔD^T, ΔD, and ΔE to increase (decrease) while making ΔS decrease (increase).*

We compute D^T, D, S and E using the simulation-based approach described in §IV and parameters listed in Table I. When changing inactivity timer values or using a new technique for tail removal, we are interested in relative changes of D^T, D, S and E compared to the default setting where we use the default state machine parameters (Table I) without removing tails. Let D_1^T and D_0^T be the DCH tail time in the new setting and in the default setting, respectively. The relative change of D^T, denoted as ΔD^T, is computed by $\Delta D^T = (D_1^T - D_0^T)/D_0^T$. We have similar definitions for ΔD, ΔS and ΔE, which will be revisited in §VI-C and §VII.

C. Fast Dormancy

The fundamental reason why inactivity timers are necessary is that the network has no easy way of predicting the network idle time of a UE. Therefore the RNC conservatively appends a tail to *every* network usage period. This naturally gives rise to the idea of letting UE applications determine the end of a network usage period since they can make use of application knowledge useful for predicting network activities. Once an imminent tail is predicted, a UE notifies the RNC, which then immediately releases allocated resources.

Based on this simple intuition, a feature called *fast dormancy* has been proposed to be included in 3GPP Release 7 [2] and Release 8 [3]. The UE sends an RRC message, which we call the T message, to the RNC through the control channel. Upon the reception of a T message, the RNC releases the RRC connection and lets the UE go to IDLE (or to a hibernating state that has lower but still non-trivial promotion delay). This feature is supported by several handsets [3]. To the best of our knowledge, no smartphone application uses fast dormancy in practice, partly due to a lack of the OS support that provides a simple programming interface.

However, based on measuring the device power consumption, we do observe that a few phones (*e.g.,* Google Nexus One) adopt fast dormancy in an *application-agnostic* manner: the UE goes to IDLE faster than other phones do for the same carrier. In other words, they use a shorter inactivity timer controlled by the device in order to lengthen the battery life. The disadvantage of such an approach is well understood [4]: the additionally incurred state promotions may introduce significant processing overheads at the RNC and may worsen user experience.

VI. TAIL OPTIMIZATION PROTOCOL

In this section, we describe our proposed Tail Optimization Protocol (TOP), an application-layer protocol that leverages the support of fast dormancy to remove tails. In TOP, applications define *data transfers* and predict the inter-transfer time at the end of each data transfer (§VI-A) using a simple interface described in §VI-B. If the predicted inter-transfer time is greater than a *tail threshold*, the application informs the RNC to initiate fast dormancy. We discuss how to set the tail threshold in §VI-C and describe how TOP handles concurrent network activities in §VI-D.

Our design of TOP requires small changes at UE applications (and optionally server applications, as a server may provide a UE with hints about predicting a tail) and the UE OS, but no change at a UE's firmware/hardware given that fast dormancy is widely deployed. Also TOP is transparent to the UTRAN and CN. Therefore TOP is incrementally deployable. Note that the T message is already supported by the RNC [2].

A. Feasibility of Tail Prediction

From applications' perspective, tail eliminations are performed for each *data transfer*, defined by applications to capture a network usage period. For example, a data transfer can correspond to all packets belonging to the same HTML page. To use TOP, an application only needs to *(i)* ensure that the current data transfer has ended, and *(ii)* provide TOP with its predicted delay between the current and the next data transfer, denoted as the *inter-transfer time* (ITT), via a simple interface described in §VI-B. ITT is essentially the packet inter-arrival time between the last packet of a transfer and the first packet of the next transfer. Note that downlink (DL) and uplink (UL) packets are not differentiated as both use the same state machine.

We first consider the most simple scenario with no concurrent network activities. TOP sends a T message (*i.e.,* invoking fast dormancy) to eliminate the tail if the predicted ITT is longer than a threshold called *Tail Threshold* (TT). A large value of TT limits the radio resource and energy savings achieved by TOP while a small TT incurs extra state promotions. We justify how we choose TT in §VI-C.

Clearly, the ITT prediction is application specific. It is easier to predict for applications with regular traffic patterns, with limited or no user interaction (*e.g.,* video streaming), but it is more difficult for user-interactive applications such as Web browsing and Google Map, as user behaviors inject randomness to the packet timing. For example, in Web browsing, each transfer corresponds to downloading one HTML page with all embedded objects. The browser knows exactly when the page has been fully downloaded. However, the timing gap between two consecutive transfers may be shorter than the tail threshold (*e.g.,* a user can quickly navigate between pages). Thus the browser should *selectively* invoke TOP. The second example is multimedia streaming. A streaming transfer consists of a single burst of packets of video/audio content (§VII-B1). The application usually can predict termination of a streaming burst. TOP can be applied if the timing gap between two consecutive bursts (usually known by the application) is longer than TT. As another example, interactive map applications involve continuous user interactions, thus TOP may not be applicable as it is very hard to define a transfer.

There are two issues related to ITT prediction. First, applications may not predict ITT accurately: misprediction can lead to increased promotion overhead due to predicting a short ITT less than TT to be a long ITT greater than TT, or lead to missing opportunities for tail removal due to predicting a long ITT to be short. A comprehensive study of prediction methodologies for interactive applications such as Web browsing is beyond the scope of this paper and is our ongoing work. Here we assume ITTs are predicted with a reasonable accuracy (*e.g.,* 80% to 90%).

The second issue is that, the existence of concurrently running applications and independent components of the same application (*e.g.,* a streaming application with an advertisement bar embedded) further complicates tail prediction. Clearly applications cannot predict other applications' concurrent network activities that affect state transitions. But there is a need to look across applications when deciding whether to invoke fast dormancy. TOP is responsible for handling the concurrency as will be described in §VI-D.

B. The Interface for Tail Removal

Unlike applications, TOP is unaware of the way applications define their transfers. TOP instead schedules tail removal requests at the *connection* level. A connection is defined as usual by five tuples: srcIP, dstIP, srcPort, dstPort, and protocol (TCP/UDP). Note that it is possible that either one connection contains multiple transfers or one transfer involves multiple connections. At the end of a transfer, after the last packet is transmitted, an application informs TOP via a simple API `TerminateTail(`c`, `δ`)` that the predicted ITT of connection c is δ. In other words, the next UL/DL packet of connection c belongs to the next transfer and will arrive after δ time units.

When user interactions are involved, it may be difficult for applications to predict the exact value of ITT. An application can then performs *binary* prediction *i.e.,* whether $ITT \leq TT$ or $ITT > TT$. The API is only called in the latter case: if ITT is predicted to be greater than TT, then `TerminateTail(`c`, `δ`)` is invoked with δ set to a fixed large value (*e.g.,* 60 sec). In fact, when no concurrent network activities exist, the exact prediction of ITT is not necessary at all as long as the binary prediction is correct. On the other hand, when concurrency exists, the predicted ITT value may affect how fast dormancy is invoked. Let the real ITT be δ_0 and the predicted ITT be δ. Underestimating δ_0 ($\delta < \delta_0$) may prevent other concurrent applications from invoking fast dormancy and overestimating δ_0 ($\delta > \delta_0$) may incur additional state promotions. However, based on our empirical evaluation using real cellular traces in §VII, we found that as long as the binary prediction is correct, the actual prediction value of ITT is much less important.

Calling `TerminateTail(`c`, `δ`)` indicates that the next transfer belongs to an *established* connection c. Also an
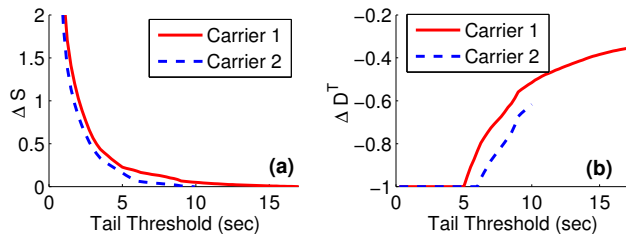
Fig. 5. Impact of Tail Threshold (TT) on (a) ΔS (b) ΔD^T

application may start the next transfer by establishing a *new* connection that does not exist when `TerminateTail` is called. For example, a Web browser may use a new TCP connection for fetching a new page. Suppose that at the end of a connection c, the application makes a prediction that the next transfer initiates a new connection and the ITT is δ. In that case, the application needs to make two consecutive calls: *(i)* `TerminateTail`($null$, δ), indicating that a new connection will be established after δ time. The first parameter is $null$ as the application does not know about the future connection. *(ii)* `TerminateTail`(c, ∞), indicating the termination of c and the termination of the current transfer. Note that the two calls trigger at most one T message. We explain why the second call is necessary in §VI-D.

C. Determining the Tail Threshold Value

A *tail threshold (TT)* is used by TOP to determine whether to send a T message when `TerminateTail` is called. When no concurrency exists, TOP sends a T message if and only if the predicted ITT is greater than TT. A large value of TT may limit radio resource and energy savings, and a small TT may incur extra state promotions.

Usually a UE is at DCH when a transfer ends. Assuming this, a T message triggers a DCH→IDLE demotion. However, in the current state machine setting, a UE will experience two state demotions: DCH→FACH with duration of α, and FACH→IDLE with duration of β. Therefore, the default value of TT should be $\alpha+\beta$ to match the original behavior. In other words, assuming the predicted ITT is δ, to ensure no additional promotion occurs (if predictions are correct), TOP should not send a T message unless $\delta > \alpha + \beta = TT$. However, such a default value of TT is large (17 sec for Carrier 1 and 10 sec for Carrier 2), limiting the effectiveness of TOP.

The key observation here is based on our empirical measurement shown in Figure 5(a), which is generated as follows. We replay calibrated sessions (§IV) against both carriers' state machines with different TT values, assuming that a T message is sent whenever the packet inter-arrival time is greater than TT. We measure the change of the total duration of state promotions ΔS (§V-B). As expected, ΔS monotonically decreases as TT increases, and reaches zero when $TT = \alpha+\beta$. Also we empirically observe that reducing TT to the value of α incurs limited promotion overhead: 22% and 7% for Carrier 1 and Carrier 2, respectively. Note that given a fixed TT, Carrier 1 has a higher promotion overhead because it has a much longer β timer. On the other hand, as shown in Figure 5(b), which plots the relationship between TT and

the change of the total DCH tail time ΔD^T, setting TT to $\alpha + \beta$ saves only 35% and 61% of the DCH tail time for the two carriers, respectively, but reducing TT to α eliminates all DCH tails (*i.e.,* $\Delta D^T = -1$). Therefore we set TT to α since it better balances the tradeoff described in §V-B.

D. Handling concurrent network activities

Tail elimination is performed for each data transfer determined by the application. As described in §VI-A, an application only ensures that the delay between consecutive transfers is longer than the tail threshold, without considering other applications. However, RRC state transitions are determined by the *aggregated* traffic of all applications. Therefore, allowing every application to send T messages independently causes problems. For example, at time t_1, TOP sends a T message for Application 1 to cut its tail. But a packet transmitted by Application 2 at t_2 will trigger an unnecessary promotion if $t_2 - t_1 < TT$.

Ideally, if all connections can precisely predict the *packet* inter-arrival time, then there exists an optimal algorithm to determine whether to send the T message. The algorithm aggregates prediction across connections by effectively treating all network activities as part of the same connection, so that fast dormancy is triggered only when the combined ITT exceeds TT. At a given time t, let $a_1, ..., a_n$ be the predicted arrival time of the next packet for each connection, then TOP should send a T packet if $\min\{a_i\} - t > TT$.

In practice, however, TOP faces two challenges. First, as mentioned in §VI-A, applications perform predictions at *transfer* level. Therefore no prediction information is available except for the last packet of a transfer. This may incur additional promotions if, for example, Connection c_1 invokes fast dormancy when Connection c_2 is in the middle of a transfer. Second, legacy applications are unaware of TOP and some applications may not use TOP due to their particular traffic patterns. To handle both issues, we design a simple and robust coordination algorithm described below.

The algorithm considers two cases to determine whether to send a T message for fast dormancy. First, for all connections *with* ITT prediction information, that is, `TerminateTail` is called but the next packet has not yet arrived, fast dormancy is triggered only when the combined ITT exceeds the tail threshold. Second, we apply a simple heuristic to handle connections *without* ITT being predicted, because either those connections are not at the end of a transfer (`TerminateTail` is not called after transmitting a packet in the middle of a transfer) or they do not use TOP (`TerminateTail` is never called). If such connections exist, a T message is not sent if any of them has recent packet transmission activity within the past p seconds where p is a predefined parameter, as for an active connection, a recent packet transmission usually indicates another packet will be transmitted in the near future. Not sending a T message at such a case reduces additional promotions. We set p to α based on our empirical measurement similar to the one described in §VI-C.

We now describe the coordination algorithm in detail by referring to the pseudo code listed in Figure 6. The algorithm

```
01 struct CONNECTION {//per-conn. states maintained by TOP
02    TIME_STAMP predict;
03    TIME_STAMP ts;
04    BOOLEAN dummy; //false for any existing connection
05 };
06 TerminateTail(CONNECTION c, ITT δ) {
07    foreach conn in Connections{ //handle out-of-date predictions
08      if (conn.predict < ts_cur) { //ts_cur is the current timestamp
09        if (conn.dummy = true)
10          {Connections.remove(conn);}
11        else {conn.predict ← null;}
12      }
13    }
14    if (c = null) { //create a dummy connection established soon
15      c ← new CONNECTION;
16      c.dummy ← true;
17      Connections.add(c);
18    }
19    c.predict ← ts_cur + δ; //update the prediction
20    foreach c' in Connections{ //check the two constraints
21      if ((c'.predict ≠ null && c'.predict < ts_cur + α)
22        || (c'.predict = null && c'.ts > ts_cur − α))
23      {return;} //fast dormancy is not invoked
24    }
25    send T message;
26 }
27 NewPacketArrival(CONNECTION c) {
28    c.ts = ts_cur;
29    c.predict ← null;
30 }
```

Fig. 6.   The coordination algorithm of TOP

maintains three states for each connection. ts and $predict$ correspond to the timestamp of the last observed packet, and the predicted arrival time of the next packet, respectively (Line 2-3). We explain the $dummy$ state shortly. Whenever an incoming or outgoing packet of connection c arrives, $c.ts$ is updated to ts_{cur}, the current timestamp, and $c.predict$ is set to $null$, indicating that no prediction information is currently available for connection c (Line 27-30). At the end of a transfer, after the last packet is transmitted, an application calls TerminateTail(c, $δ$). Then TOP updates $c.predict$ to $ts_{cur} + δ$ (Line 19) and sends a T message if both conditions hold (Line 20-25).

$$\min_{c'}\{c'.predict \neq null\} > ts_{cur} + TT \qquad (1)$$

$$\forall c' : c'.predict = null \rightarrow c'.ts < ts_{cur} - p \qquad (2)$$

where c' goes over all connections and "$\rightarrow$" denotes implication. Equation (1) and (2) represent two aforementioned cases where connections are with and without prediction information, respectively. Note that both the tail threshold TT in Equation (1) (Line 21) and the p value in Equation (2) (Line 22) are both empirically set to α.

Recall that in §VI-B, when the next transfer starts in a new connection, an application calls TerminateTail($null$, $δ$) then TerminateTail(c, ∞) at the end of connection c, which is also the end of current transfer. TOP handles the first call by creating a dummy connection c_d (Line 14-18) with $c_d.predict = ts_{cur} + δ$, and c_d is considered in Equation (1). The dummy connection c_d is removed when its prediction

is out-of-date *i.e.*, $ts_{cur} > c_d.predict$ (Line 9-10). For an established (*i.e.*, not dummy) connection c, $c.predict$ is set to $null$ (no prediction information) when it is out-of-date (Line 11), and c is removed *i.e.*, not considered by Equation (1) or (2), when c is closed.

An application may call TerminateTail($null$, $δ$) at ts_{cur}, *immediately* after the last packet of connection c is transmitted. However, it is possible that at ts_{cur}, c is not yet removed by TOP although no packet of c will appear. In this case, $c.ts$, the timestamp of the last packet of c, is very close to ts_{cur}, making Equation (2) not hold. Thus a T message will never be sent. The problem is addressed by the second call TerminateTail(c, ∞) that sets $c.predict = \infty$. Therefore making two calls guarantees that a T message is properly sent even if c is not timely removed.

An application abusing fast dormancy can make a UE send a large amount of T messages, each of which may cause a state demotion to IDLE followed by a promotion triggered by a packet, in a short period. To prevent such a pathological case, TOP sends at most one T message for every t seconds even if multiple T messages are allowed by the constraints of Equation (1) and (2)(not shown in the pseudo code). This guarantees that repeatedly calling TerminateTail is harmless, and that the frequency of the additional state promotions caused by TOP is no more than one per t seconds. We empirically found that setting t to 6 to 10 seconds has negligible adverse impact on resource savings for normal usage of TOP.

We notice that the major runtime overhead of TOP is to intercept packets and to record their timestamps (Line 27-30). We implemented a kernel module for that task on an Android G2 smartphone. By measuring the additional CPU utilization, we found that the runtime overhead is negligible regardless of the network throughput.

VII. EVALUATIONS

We use real traces to demonstrate radio resource and energy savings brought by TOP, focusing on evaluating how well TOP handles concurrent network activities. In §VII-A, we use the passive trace described in §IV to study the impact of TOP on a large number of users. In §VII-B, we perform case studies of two applications using traces locally collected by Tcpdump from an Android G2 phone.

We use ΔD, ΔD^T, ΔE, and ΔS defined in §V-B as evaluation metrics. They are computed using the simulation-based approach described in §IV. The comparison baseline is the default state machine configuration without using a tail removal technique for the same carrier. For TOP, we set both TT (§VI-C) and p (§VI-D) to the α timer value. TOP sends at most one T message for every $t = 10$ seconds.

A. Evaluation using passive traces

The evaluation is performed at a per-session basis using the calibrated passive trace described in §IV. For each session, we extract connections (defined by 5-tuples) and classify them into four types by the port number of UE's TCP peer, since only TCP headers are available: Web (80, 443, 8080), Email (993, 995, 143, 110, 25), Sync (a popular synchronization

service of Carrier 1 using a special port number), and Other (all other port numbers). They contribute to 78.8%, 15.1%, 0.2%, and 5.9% of the total traffic volume, respectively. We use a threshold of 10 sec of idle time to decide that a connection has terminated. Changing this value does not qualitatively affect the simulation results.

For simplicity, we assume that there are four applications, each involving one traffic type, running on smartphones. For Web, Email, and Sync applications, a transfer is defined as consecutive connections of the same traffic type whose inter-connection time (the interval between the last packet of one connection and the first packet of the next connection) is less than 1 sec. Note that a transfer may consist of multiple connections and connections may overlap (*e.g.,* concurrent connections supported by smartphone browsers). At the end of each transfer, each application independently calls `TerminateTail` with probability of z, which quantifies the applicability of TOP, to perform binary predictions (whether ITT is greater than TT, see §VI-B) with accuracy of w. In other words, the probabilities of a correct prediction, an incorrect prediction, and no prediction (`TerminateTail` is not invoked) are zw, $z(1-w)$, and $1-z$, respectively. Since binary predictions are performed, each application uses an ITT of 60 sec if the ITT is predicted to be greater than TT. Varying this from 30 sec to infinity, or using the exact prediction value of ITT changes the results in Figure 7 by no more than 0.01. We assume that the "Other" application is unaware of TOP.

Figure 7 plots the impact of TOP on ΔE, ΔS, ΔD^T, and ΔD by varying w and z for Carrier 1. In each plot, the $z = 0$ curve is a horizontal line at $y = 0$ corresponding to the comparison baseline *i.e.,* the default case where TOP or fast dormancy is not used. Figure 7 clearly shows that, increasing z, the applicability of TOP, brings more savings at the cost of increasing the state promotion delay. On the other hand, increasing w, the prediction accuracy, not only benefits resource savings but also reduces the state promotion overhead. Under the case where $z = 0.8$ and $w = 90\%$, TOP saves the overall radio energy E, the DCH tail time D^T, and the total DCH time D by 17.4%, 55.5%, and 11.7%, respectively, with the state promotion delay S increasing by 14.8%. The results for Carrier 2 show similar trends. Under the condition of $z = 0.8$ and $w = 90\%$, TOP can save E, D^T, and D by 14.9%, 60.1%, and 14.3%, respectively with S increasing by 9.0%.

We compare TOP with other schemes for saving the tail time. In each plot of Figure 8, the X axis is the state promotion delay ΔS, and the Y axis corresponds to saved resources (ΔE, ΔD^T, or ΔD) for Carrier 2. A more downward or leftward curve indicates a better saving scheme since given a fixed ΔS, we prefer a more negative value of ΔE, ΔD^T, or ΔD indicating higher resource savings. Each plot of Figure 8 contains four curves. The "TOP" curve corresponds to using TOP with $w = 80\%$ and z being varied from 0.5 to 1.0. The "FD" (fast dormancy) curve is generated using the same parameters, but in the "FD" scheme, applications use fast dormancy without being scheduled by TOP. In other words,

an application (Web, Email, or Sync) sends a $\boldsymbol{T}$ message whenever its predicted ITT is greater than TT. The "timer" curve corresponds to a strategy of proportionally decreasing α and β timers that affect all sessions in the trace.

The "TE" curve denotes employing TailEnder [6] to save energy and radio resources. As described in §I, for delay-tolerant applications, their data transfers can be delayed and batched to reduce the tail time. TailEnder is a scheduling algorithm that schedules transfers to minimize the energy consumption while meeting user-specified deadlines by delaying transfers and transmitting them together. The TailEnder algorithm was implemented in our simulator using the default parameters described in [6]. We apply TailEnder on all Email and Sync transfers and vary the deadline (the maximally tolerated delay) from 0 to 5 minutes. A longer deadline can potentially save more resources but a user has to wait for longer time.

We discuss the results in Figure 8. TOP outperforms fast-dormancy (FD), whose curve lies on the right of the "TOP" curve. To achieve the same savings in D, E, and D^T, the state promotion delay of TOP is always less than that of FD by 10% of the overall promotion delay in the default scheme. Further, reducing inactivity timers incurs additional state promotions, overwhelming the savings of D and E. The fundamental reason for this is the static nature of the inactivity timer paradigm where all packets experience the same timeout period. We also notice that TailEnder can reduce the overall state promotion delay (as indicated by the negative ΔS values) due to its batching strategy. However, its applicability is very limited, yielding much less savings, and it incurs additional waiting time for users. The comparison results for Carrier 1 is qualitatively similar, implying that invoking fast dormancy with a reasonable prediction accuracy (around 80%) surpasses the traditional approach of tuning inactivity timers in balancing the tradeoff, and TOP's coordination algorithm effectively reduces the state promotion overhead caused by concurrent network activities.

B. Evaluation using locally collected traces

We perform case studies of two applications (Pandora streaming and Web browsing) using traces locally collected from an Android G2 phone using Carrier 2's UMTS network. We investigate each application separately without injecting concurrent traffic, then apply the coordination algorithm on the aggregated traffic of both applications.

1) Pandora radio streaming: Pandora [1] is an Internet radio application. We collected a 30-min trace using Tcpdump by logging onto one author's Pandora account, selecting a pre-defined radio station, then listening to seven tracks (songs). By analyzing the trace, we found that the Pandora traffic consists of two components: the audio/control traffic and the advertisement traffic. Before a track is over, the content of the next track is transferred in one burst utilizing the maximal bandwidth. Then at the exact moment of switching to the next track, a small traffic burst of control messages is generated. The second component is periodical advertisement traffic from an Amazon EC2 server for every one minute. Each such burst can trigger an IDLE→DCH promotion.
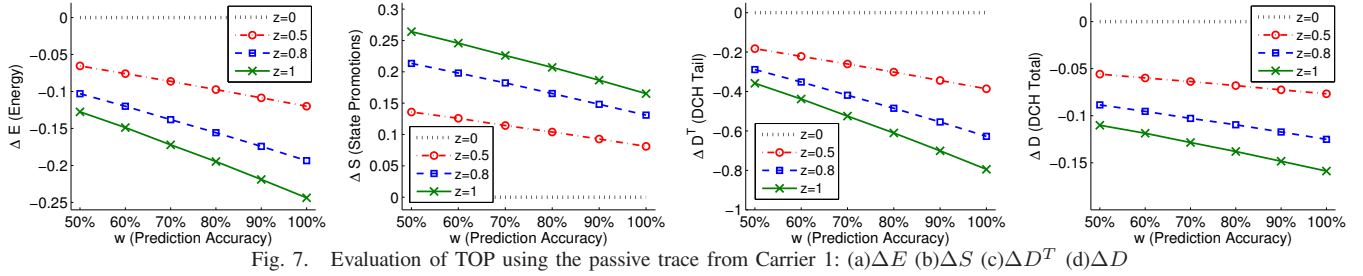
Fig. 7. Evaluation of TOP using the passive trace from Carrier 1: (a)ΔE (b)ΔS (c)ΔD^T (d)ΔD
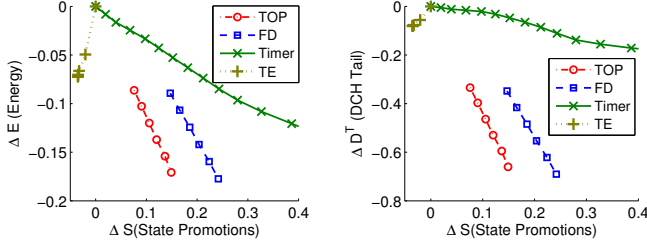

Fig. 8. Comparison of four schemes of saving tail time for Carrier 2: (a)ΔS vs. ΔE (b)ΔS vs. ΔD^T (c)ΔS vs. ΔD

TABLE II
IMPACT OF TOP ON PANDORA

ΔD^T	-100%
ΔD	-50.1%
ΔS	3%
ΔE	-59.2%

We apply TOP on the trace by regarding each data/control burst and each advertisement burst as a transfer. The results in Table II indicate that TOP achieves remarkably good resource savings for traffic patterns consisting of bursts separated by long timing gaps. TOP eliminates all tails that contribute to about half of the total DCH time with the state promotion delay increasing by 3% regardless of using binary or exact prediction of ITT. The radio energy usage decreases by 59%.

2) Web Browsing: We show the applicability of TOP to Web browsing. As described in §VI-A, here a transfer can be naturally defined as packets belonging to the same Web page (including embedded objects) downloaded in a burst. Usually a browser can precisely know the termination of a transfer, and the challenging part is to predict inter-transfer times (ITTs) that involve user interactions, as a page download is mostly triggered by clicking a link, and ITTs correspond to a user's reading or thinking time. A comprehensive study of the prediction methodology is beyond the scope of this paper. Here we describe our preliminary study showing that even very simple heuristics can lead to good prediction results for some popular websites.

We simultaneously collected user input event traces (*e.g.,* tapping the screen) and packet traces from eight users while they visited the three websites listed in Table III. The traces we collected have a total duration of 298 minutes with each between 7 and 20 minutes long. Then we extract individual transfers by examining HTTP headers and by correlating packet traces with user input events. We found that all transfers were triggered by users *i.e.,* a user input is observed within 0.5s (to tolerate the processing delay) before a transfer starts.

Figure 9 plots the CDF of ITTs for the three websites. Each curve consists of ITTs of all eight users. We call an ITT whose value is greater than TT, which is 6 sec, a long ITT. Otherwise it is a short ITT. Ideally a T message should only be sent for a long ITT. Figure 9 clearly indicates the wide disparity of traffic patterns among websites due to their different contents. For cnn and amazon, 92% and 91% of ITTs are long. In
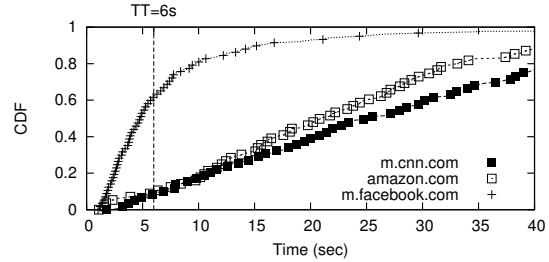

Fig. 9. CDF of inter-transfer time (ITT) for three websites

TABLE III
IMPACT OF TOP ON TRACES OF THREE WEBSITES

Website	m.cnn.com	amazon.com	m.facebook.com
% long ITT	91.2±4.8%	91.4±6.5%	38.1±5.9%
ΔD^T	-90.7%	-88.7%	-36.7%
ΔD	-51.4%	-48.0%	-22.2%
ΔE	-61.1%	-57.4%	-21.6%
ΔS	+8.4%	+9.8%	+51.5%

contrast, facebook has much less long ITTs (only 38%). The second row of Table III shows the average ratio of long ITTs for each user. The small standard deviations indicate that the long-ITT ratio is relatively stable among the eight users.

Figure 9 suggests that a browser can adopt a simple approach where for websites with historically observed high long-ITT ratios, the browser always predicts an ITT to be long by setting ITT to a large value (we use 60 sec, assuming browsers cannot predict the exact values of ITT and do only binary predictions). The simulation results are shown in Table III. For cnn and amazon, TOP saves about half of the total DCH time and 60% of the radio energy with less than 10% of increase on the state promotion overhead. However, for facebook, the savings are much less and the promotion overhead becomes high due to its small long-ITT ratio. The browser can thus use a fixed long-ITT ratio threshold for deciding whether or not to apply TOP for each website.

3) Evaluation of mixed traces: To evaluate the coordination algorithm, for cnn and amazon, we concatenate traces of 2

TABLE IV
IMPACT OF TOP ON MIXED TRACES (PANDORA AND CNN)

Pandora+CNN	ΔD^T	ΔD	ΔS	ΔE
Default	0	0	0	0
TOP	-88±3%	-41±2%	+17±3%	-48±2%
FD	-100±0%	-50±2%	+66±9%	-58±2%

to 4 randomly selected users, then mix the concatenated trace (roughly 30 min) with the 30-min Pandora trace. We assume that three application components concurrently use TOP: Pandora audio (exact prediction of ITT), Pandora advertisement (exact prediction), and the browser application (always predict ITT to be long). For each website, we generate 100 mixed traces and feed each of them into the simulator for the three schemes listed in Table IV: "Default" (the comparison baseline where TOP or fast dormancy is not used), "TOP" (using TOP), and "FD" (only using fast dormancy). Table IV (`Pandora + cnn`) clearly shows the benefits of TOP, which significantly decreases ΔS from 66% to 17% by reasonably sacrificing savings of other three dimensions. The results for `Pandora + amazon` are very similar.

VIII. RELATED WORK

With the increasing number of 3G network users, radio resource management has become a critical topic for both academia and industry. Much effort has been put on the study of the inactivity timers used in radio resource release. These studies can be classified into two categories, those that attempt to determine optimal inactivity timer values by doing theoretical analysis and simulation, and those that study the impact of the deployed timers on smartphone energy consumption and network performance. There exists previous work for selecting optimal inactivity timer values, but most are based on particular traffic models. In [8] the impact of inactivity timers on the UMTS network capacity was studied by simulating the performance of web browsing. [14], [23] proposed analytical models to measure the energy consumption of user device under different timer values. [15], [21] also discussed the influence of different timeout values on both service quality and energy consumption. In addition, several other projects studied network resource management [10], [20]. Distinct from finding the optimal inactivity timer values, Liers *et al.* [16] proposed to decide timeouts dynamically and specifically based on the current load, radio and code resources, and processing resources. Based on the deployed inactivity timers in current commercial networks, [22], [11], [17] carried out measurement studies to examine the energy consumption and network performance on smartphones. [5] attempted to optimize network performance and increase energy efficiency. To save the energy of UEs by requiring less radio resources, researchers have proposed to shift the traffic pattern to adapt to the existing timers, such as TailEnder proposed by [6].

IX. CONCLUDING REMARKS

By leveraging fast dormancy, TOP enables applications to actively inform the network of a tail that can be eliminated via a simple interface. Our design of TOP enables significantly better radio resource usage and substantial energy savings for cellular networks. More importantly, our work opens new research opportunities for designing effective tail prediction algorithms for smartphone applications (especially for applications involving user interactions), which is the major part of our ongoing work. In addition, we are seeking ways to build a real implementation of TOP on Android phones.

REFERENCES

[1] Pandora Radio. http://www.pandora.com.

[2] UE "Fast Dormancy" behavior. 3GPP discussion and decision notes R2-075251, 2007.

[3] Configuration of fast dormancy in release 8. 3GPP discussion and decision notes RP-090960, 2009.

[4] System impact of poor proprietary fast dormancy. 3GPP discussion and decision notes RP-090941, 2009.

[5] M. Anand, E. B. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. *Wireless Networks*, 11(4), 2005.

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

[7] M. Chatterjee and S. K. Das. Optimal mac state switching for cdma2000 networks. In *INFOCOM*, 2002.

[8] M. Chuah, W. Luo, and X. Zhang. Impacts of Inactivity Timer Values on UMTS System Capacity. In *Wireless Communications and Networking Conference*, 2002.

[9] Q. Engineering Services Group. System Parameter Recommendations to Optimize PS Data User Experience and UE Battery Life, 2007.

[10] M. Ghaderi, A. Sridharan, H. Zang, D. Towsley, and R. Cruz. Tcp-aware resource allocation in cdma networks. In *Proceedings of ACM MOBICOM*, Los Angeles, CA, USA, September 2006.

[11] H. Haverinen, J. Siren, and P. Eronen. Energy Consumption of Always-On Applications in WCDMA Networks. In *Proceedings of IEEE Vehicular Technology Conference*, 2007.

[12] H. Holma and A. Toskala. HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications. John Wiley and Sons, Inc., 2006.

[13] H. Kaaranen, A. Ahtiainen, L. Laitinen, S. Naghian, and V. Niemi. UMTS Networks: Architecture, Mobility and Services (2nd Edition). John Wiley and Sons, Inc., 2005.

[14] C.-C. Lee, J.-H. Yeh, and J.-C. Chen. Impact of inactivity timer on energy consumption in WCDMA and cdma2000. In *Wireless Telecommunications Symposium*, 2004.

[15] F. Liers, C. Burkhardt, and A. Mitschele-Thiel. Static RRC Timeouts for Various Traffic Scenarios. In *PIMRC*, 2007.

[16] F. Liers and A. Mitschele-Thiel. UMTS data capacity improvements employing dynamic RRC timeouts. In *PIMRC*, 2005.

[17] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G network: interplay between the wireless channel and applications. In *Mobicom*, 2008.

[18] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *IMC*, 2010.

[19] S. Sesia, I. Toufik, and M. Baker. LTE: The UMTS Long Term Evolution From Theory to Practice. John Wiley and Sons, Inc., 2009.

[20] A. Sridharan, R. Subbaraman, and R. Guerin. Distributed Uplink Scheduling in CDMA Networks. In *Proceedings of IFIP-Networking 2007*, May 2007.

[21] A. Talukdar and M. Cudak. Radio resource control protocol configuration for optimum Web browsing. In *Proceedings IEEE 56th Vehicular Technology Conference*, 2002.

[22] W. L. Tan and O. Yue. Measurement-based Performance Model of IP Traffic over 3G Networks. *TENCON 2005 IEEE Region 10*.

[23] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. Comparative Analysis of Energy-Saving Techniques in 3GPP and 3GPP2 Systems. *IEEE transactions on vehicular technology*, 58(1):432–438, January 2009.