Traffic-Aware Techniques to Reduce 3G/LTE Wireless Energy Consumption

Shuo Deng and Hari Balakrishnan
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, USA
shuodeng@csail.mit.edu, hari@csail.mit.edu

ABSTRACT

The 3G/LTE wireless interface is a significant contributor to battery drain on mobile devices. A large portion of the energy is consumed by unnecessarily keeping the mobile device's radio in its "Active" mode even when there is no traffic. This paper describes the design of methods to reduce this portion of energy consumption by learning the traffic patterns and predicting when a burst of traffic will start or end. We develop a technique to determine when to change the radio's state from Active to Idle, and another to change the radio's state from Idle to Active. In evaluating the methods on real usage data from 9 users over 28 total days on four different carriers, we find that the energy savings range between 51% and 66% across the carriers for 3G, and is 67% on the Verizon LTE network. When allowing for delays of a few seconds (acceptable for background applications), the energy savings increase to between 62% and 75% for 3G, and 71% for LTE. The increased delays reduce the number of state switches to be the same as in current networks with existing inactivity timers.

CATEGORIES AND SUBJECT DESCRIPTORS

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.4 [**Performance of Systems**]: Performance attributes; Design studies

GENERAL TERMS

Design, Performance

KEYWORDS

Cellular Networks, Energy Saving

1. INTRODUCTION

Over a fifth of the 5.5 billion active mobile phones today have "broadband" data service, and this fraction is rapidly growing. Smartphones and tablets with wide-area cellular connectivity have become a significant, and in many cases, dominant, mode of network access. Improvements in the quality of such network connectivity suggest that mobile Internet access will soon overtake desktop access, especially with the continued proliferation of 3G networks and the emergence of LTE and 4G.

Wide-area cellular wireless protocols need to balance a number of conflicting goals: high throughput, low latency, low signaling overhead (signaling is caused by mobility and changes in the mobile device's state), and low battery drain. The 3GPP and 3GPP2 standards (used in 3G and LTE) provide some mechanisms for the cellular network operator and the mobile device to optimize these metrics [22, 3], but to date, deployed methods to minimize energy consumption have left a lot to be desired.

The 3G/LTE radio consumes significant amounts of energy; on the iPhone 4, for example, the stated talk time is "up to 7 hours on 3G" (i.e., when the 3G radio is on and in "typical" use) and "up to 14 hours on 2G".[1] On the Samsung Nexus S, the equivalent numbers are "up to 6 hours 40 minutes on 3G" and "up to 14 hours on 2G".[2] That the 3G/LTE interface is a battery hog is well-known to most users anecdotally and from experience, and much advice on the web and on blogs is available on how to extend the battery life of your mobile device.[3] Unfortunately, essentially all such advice says to "disable your 3G data radio" and "change your fetch data settings to reduce network usage". Such advice largely defeats the purpose of having an "always on" broadband-speed wireless device, but appears to be the best one can do in current deployments.

We show the measured values of 3G energy consumption for multiple Android applications in Figure 1.[4] This bar graph shows the percentage of energy consumed by different 3G radio states. For most of these applications (which are all background applications that can generate traffic without user input, except for Facebook), less than 30% of the energy consumed was during the actual transmission or reception of data. Previous research arrived at a similar conclusion [4]: about 60% of the energy consumed by the 3G interface is spent when the radio is not transmitting or receiving data.

In principle, one might imagine that simply turning the radio off or switching it to a low-power idle state is all it takes to reduce energy consumption. This approach does not work for three reasons. First, switching between the active and the different idle states takes a few seconds because it involves communication with the base station, so it should be done only if there is good reason to believe that making the transition is useful for a reasonable duration of time in the future. Second, switching states consumes energy, which means that if done without care, overall energy consumption will increase compared to not doing anything at all. Third, the switching incurs signaling overhead on the wireless network, which means

CoNEXT'12, December 10–13, 2012, Nice, France.

[1] http://www.apple.com/iphone/specs.html
[2] http://www.gsmarena.com/samsung_google_nexus_s-3620.php
[3] http://www.intomobile.com/2008/07/23/extend-your-iphone-3gs-battery-life/
[4] An HTC G1 phone connected to a power monitor [13], with only one application running, at one indoor location.

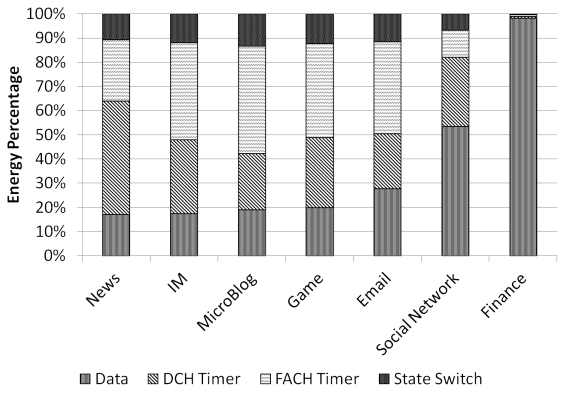


Figure 1: Energy consumed by the 3G interface. "Data" corresponds to a data transmission; "DCH Timer" and "FACH Timer" are each the energy consumed with the radio in the idle states specified by the two timers, and "State Switch" is the energy consumed in switching states. These timers and state switches are described in §2.

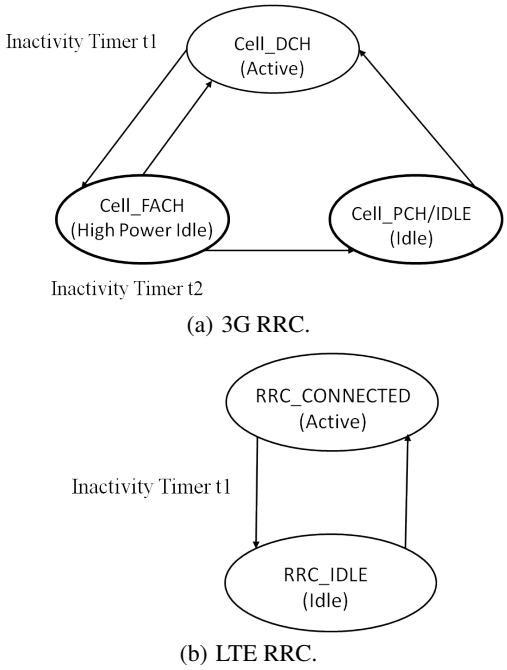


Figure 2: Radio Resource Control (RRC) State Machine.

that it should be done only if the benefits are substantial relative to the cost on the network.

This paper tackles these challenges and develops a solution to reduce 3G/LTE energy consumption without appreciably degrading application performance or introducing a significant amount of signaling overhead on the network. Unlike currently deployed methods that simply switch between radio states after fixed time intervals—an approach known to be rather crude and sub-optimal [21, 4, 11, 19])– our approach is to observe network traffic activity on the mobile device and switch between the different radio states by adapting to the workload.

The key idea is that by observing network traffic activity, a *control module* on the mobile device can adapt the 3G/LTE radio state transitions to the workload. We apply statistical machine learning techniques to predict network activity and make transitions that are suggested by the statistical models. This approach is well-suited to the emerging *fast dormancy* mechanism [1, 2] that allows a radio to rapidly move between the Active and Idle states and vice versa. Our goal is to reduce the energy consumed by networked background applications on mobile devices.

This paper makes the following contributions:

1. A traffic-aware design to control the state transitions of a 3G/LTE radio taking energy consumption, latency, and signaling overhead into consideration. The design incorporates two algorithms:
   (a) MakeIdle, which uses aggregate traffic activity to predict the end of an active session by building a conditional probability distribution of network activity.
   (b) MakeActive, which delays the start of a new session by a few seconds to allow multiple sessions to all become active at the same time and therefore reduce signaling overhead. This method is appropriate for non-interactive background applications that can tolerate some delay.
2. An experimental evaluation of these methods on real usage data from nine users over 28 total days on four different carriers. We find that the energy savings compared to the status quo range between 51% and 66% across the carriers for 3G, and is 67% on the Verizon LTE network. When allowing for delays of a few seconds (acceptable for background applications), the energy savings increase to between 62% and 75% for 3G, and 71% for LTE. The increased delays reduce the number of state switches to be the same as in current networks with existing inactivity timers.

2. BACKGROUND

This section describes the 3G/LTE state machine and its energy consumption.

2.1 3G/LTE State Machine

The Radio Resource Control (RRC) protocol, which is part of the 3GPP standard, incorporates the state machine for energy management shown in Figure 2.

The base station maintains two inactivity timers, t_1 and t_2, for each mobile device. For a device maintaining a dedicated channel in the Active (Cell_DCH) state with the base station, if the base station sees no data activity to or from the device for t_1 seconds, it will switch the device from the dedicated channel to a shared low-speed channel, transitioning the device to the "High-power idle" (Cell_FACH) state. This state consumes less power than "Active", but still consumes a non-negligible amount of power. If there is no further data activity between the device and base station for another t_2 seconds, the base station will turn the device to either the Cell_PCH or IDLE state. We refer to the Cell_PCH and IDLE states together as the "Idle" state, because the device consumes essentially no power in either state. For LTE networks (Figure 2(b)), there are only two states: RRC_CONNECTED and RRC_IDLE (there are substates in RRC_CONNECTED [8], which we do not discuss here because they are not relevant), and one inactivity timer, shown as t_1.

The inactivity timers (t_1 and t_2) are useful because a state transition from "Idle" to "Active" (Cell_DCH) incurs significant delays. For example, in our measurements in the Boston area, these values are ≈ 1.4 seconds on AT&T's 3G network, ≈ 3.6 seconds on T-Mobile's 3G network, ≈ 2.0 seconds on Sprint's 3G network, ≈ 1.0 second on Sprint's LTE network, ≈ 1.2 seconds on Verizon's 3G network, and ≈ 0.6 seconds on Verizon's LTE network (these numbers may vary across different regions). Each state transition also consumes energy on the device and incurs signaling overhead for

the base station to allocate a dedicated channel to the device. The inactivity timers also prevent the base station from frequently releasing and re-allocating channels to devices which causes per-packet delay for the device to be high.

The description given above captures the salient features of the 3GPP standard. Another popular 3G standard is 3GPP2 [3]. Although 3GPP2 networks use different techniques, from the perspective of energy consumption, they are essentially identical to 3GPP [21]; like 3GPP, 3GPP2 networks also have different power levels for different states on the device side, and use similar inactivity timers for state transitions. For concreteness, in this paper, we focus on 3GPP networks.

2.2 Energy Consumption

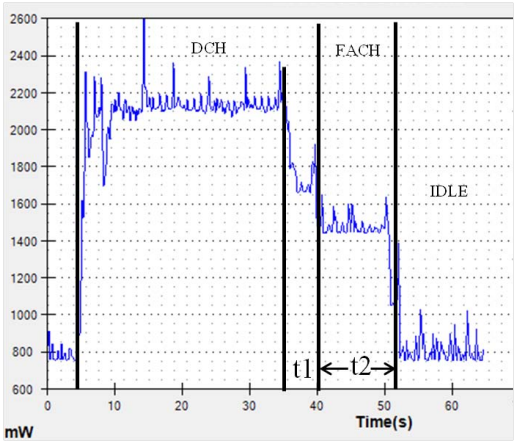
We measured the power consumption and inactivity timer values using the Monsoon Power Monitor [13]. Figure 3 shows graphs of our measurements during a radio state switches cycle on an HTC Vivid smartphone in AT&T's 3G network and on a Galaxy Nexus smartphone in Verizon's LTE network. (We show results for other carriers in Section 6.) During the High-power idle (FACH for AT&T) and part of Active (DCH for AT&T, RRC_CONNECTED for Verizon) states, there is no data transmission. The RRC state machine keeps the radio on here in case a new transmission or reception occurs in the near future. Consistent with previous work [4], we use the term *tail* to refer to this duration when the radio is on but there is no data transmission.

We measured the inactivity timer values in AT&T's 3G network in the Boston area to be $t_1 \approx 6.2$ seconds and $t_2 \approx 10.4$ seconds. The energy consumed at the end of a data transfer when the radio is in one of the two Idle states before turning off is termed the *tail energy*; this energy can be 60% or more of the total energy consumption of 3G [4].

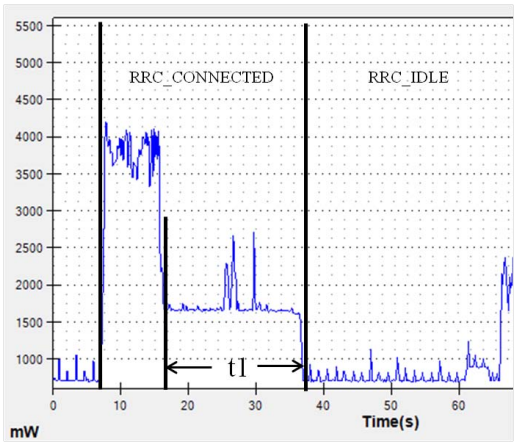
3GPP Release 7 [1] proposed a feature called *fast dormancy*, which allows the device to actively release the channel by itself before the inactivity timer times out on the base station. One of the issues that then arises is that the base station loses control over the connection when mobile devices are able to disconnect by themselves. In 3GPP Release 8 [2], fast dormancy was changed: the mobile device first sends a fast dormancy request, and the base station will decide to release the channel or not. In Europe, Nokia Siemens Networks has applied Network Controlled Fast Dormancy based on 3GPP Release 8. Because it is not entirely clear what policy any given network carrier will use to decide whether to release the channel upon receiving a request at a base station, in our simplified model, we assume that if the base station is running 3GPP Release 8, whenever the phone sends a fast dormancy request to the base station, the base station will accept and release the channel. Our goal is to evaluate the network signaling overhead of such a strategy as a way to help inform network-carrier policy.

3. DESIGN

The key insight in our approach to reduce 3G energy consumption is that by observing and adapting to network activity, a *control module* can predict when to put the radio into its Idle state, and when to move from Idle to Active state. These state transitions take a non-trivial amount of time—between 1 and 3 seconds—and also add signaling overhead because each transition is accompanied by a few messages between the device and the base station. Hence, the intuition in our approach is to predict the occurrence of *bursts* of network activity, so that the control module can put the radio into the idle mode when it believes a burst has ended, which means there will not be any more traffic in the future for a relatively long period of time. Conversely, the idea is to put the radio in active mode when "enough" bursts of traffic accumulate.



(a) HTC Vivid in AT&T 3G Network.



(b) Galaxy Nexus in Verizon LTE Network.

Figure 3: The measured power consumption of the different RRC states. Exact values can be found in Table 2. In these figures the power level for IDLE/RRC_IDLE is non-zero because of the CPU and LED screen power consumption.

To achieve the prediction, our approach needs to observe network activity and be able to pause data transmissions. To make our approach work with existing applications, we should not require any change to the application code. To achieve these goals, we modified the socket layer and added a control module inside the Android OS source code.

Our system has two software modules: one that modifies the library used by applications to communicate with the socket layer, and another that implements the control module, as shown in Figure 4. The first module informs the control module of all socket calls; in response, the control module configures the state of the radio. The fast dormancy interface is shown as a dashed module because our system uses it if it is available.

The control module implements two different methods. The first method, called MakeIdle, runs when the radio is in the Active state (Cell_DCH or RRC_CONNECTED) and determines when the radio should be put into the Idle (IDLE or Cell_PCH or RRC_IDLE) state. The second method, called MakeActive, runs when the radio is in the Idle state. In this state, it cannot send any packets without first moving to the Active state; MakeActive determines how long the radio should be idle before moving to active state.

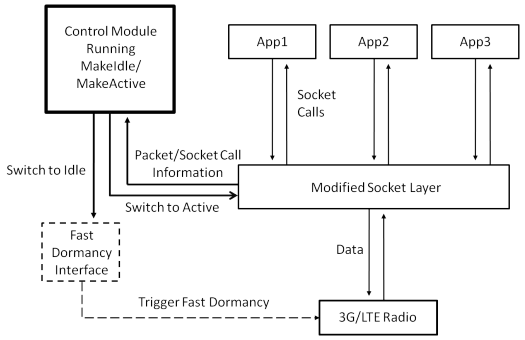


Figure 4: System design.

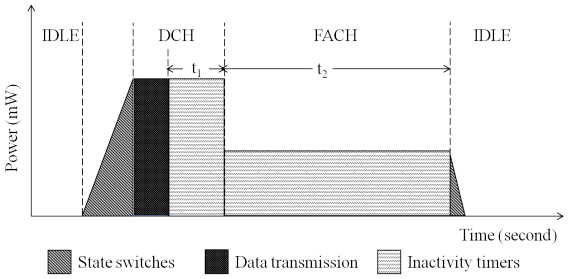


Figure 5: Simplified power model for 3G energy consumption (for an LTE model, t_2 equals to zero).

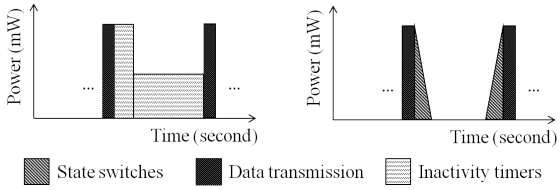


Figure 6: If the energy consumed by the picture on the right is less than the one on the left, then turning the radio to Idle soon after the first transmission will consume less energy than leaving it on. The energy is easily calculated by integrating the power profiles over time.

4. MAKEIDLE ALGORITHM

Instead of using a fixed inactivity timer, the MakeIdle method dynamically decides when to put the radio into Idle mode after each packet transmission or reception. We first show in §4.1 how to compute the optimal decision *given* complete knowledge of a packet trace: the result is that the radio should be turned to Idle if there is a gap of more than a certain threshold amount of time in the trace, which depends on measurable parameters. Then, in §4.2, we develop an online method to predict idle durations that will exceed this threshold by modeling the idle time using a conditional probability distribution.

4.1 Optimal Decision From Offline Trace Analysis

Suppose we are given a packet trace containing the timestamps of packets sent and received on a mobile device. Our goal is to determine offline when to turn the radio to the Idle state to minimize the energy consumed.

Figure 5 shows a simplified power model we use to calculate tail energy. If the inter-arrival time between two adjacent packets is t seconds, then $E(t)$, the energy consumed by the current RRC protocol with inactivity timer values t_1 and t_2 (see Figure 2), is

$$E(t) = \begin{cases} t \cdot P_{t_1} & 0 < t \le t_1 \\ t_1 \cdot P_{t_1} + (t - t_1) \cdot P_{t_2} & t_1 < t \le t_1 + t_2 \\ t_1 \cdot P_{t_1} + t_2 \cdot P_{t_2} + E_{switch} & t > t_1 + t_2 \end{cases}$$

Here, P_{t_1} and P_{t_2} are the power values for the active state and high-power idle state, respectively; the power consumed in the low-power idle state is negligible. E_{switch} is the energy consumed by switching the radio to Idle mode after the first packet transmission and then switching it back to Active for the second packet transmission. It is a fixed value for a given type of mobile device and is easy to measure.

On the other hand, if the radio switches to Idle mode immediately after the first packet transmission finishes, the energy consumed is just E_{switch}.

To minimize the energy consumed between packets, the radio should switch to Idle mode after a packet transmission if, and only if, $E_{switch} < E(t)$. Notice that because $E(t)$ is a monotonically non-decreasing function of t, there exists a value for t, which we call $t_{threshold}$, for which $E_{switch} < E(t)$ if and only if $t > t_{threshold}$. This expression quantifies the intuitive idea that after each packet, the radio should switch to Idle mode only if we know that next packet will not arrive soon; concretely, not arrive in the following $t_{threshold}$ seconds. For example, on an HTC Vivid phone in the AT&T 3G network deployed in the Boston area, $t_{threshold}$ works out to be 1.2 seconds.

4.2 Online Prediction

To minimize energy consumption in practice, we need to predict whether the next packet will arrive (to be received or to be sent) within $t_{threshold}$ seconds. Of course, we would like to make this prediction as quickly as possible, because we would then be able to switch the radio to Idle mode promptly. We make this prediction by assuming that the packet inter-arrival distribution observed in the recent past will hold in the near future. After each packet, the method waits for a short period of time and sees whether any more packets arrive. If a packet arrives, the method resets and waits, but if not, it means a transfer may be finished and the radio should switch to Idle mode.

The strategy works as follows:

1. Without loss of generality, suppose the current time is $t = 0$. Compute the conditional probability that no packet will arrive within $t_{wait} + t_{threshold}$ seconds, *given that* no packet has arrived in t_{wait} seconds.

   $$P(t_{wait}) = \mathbb{P}(\text{no packet in } t_{wait} + t_{threshold} | \text{no packet in } t_{wait})$$

   This conditional distribution is easy to compute given observations of the packet arrival times of the last several packets.
   From the traces we collected, we observed that $P(t_{wait})$ increases as t_{wait} increases, when t_{wait} is in the range of $[0, t_{threshold}]$ (if t_{wait} is greater than $t_{threshold}$, it means the radio has been idle for too long time after the packet transmission and there is not much room for energy saving). This property implies that the longer the radio waits and sees no packet, the higher the likelihood that no packet will arrive soon.
2. Now we need to find t_{wait} in order to make the likelihood "high enough". During t_{wait}, the radio consumes energy, so to decide how much is "high enough", we should take energy consump-

tion into account. Our answer is: *$P(t_{wait})$ is "high enough" if the expected energy consumption of waiting for t_{wait} and then switching states is less than the expected consumption of waiting for the inactivity timer to time out in the next t_{wait} seconds.*
The method determines t_{wait} by *minimizing* the expected energy consumption across all possible values of t_{wait}, and taking the value that minimizes the consumption. We explain how below.

The expected energy consumption of waiting for t_{wait} and then switching states is:

$$\mathbb{E}[E_{wait\_switch}] \quad = \quad [E_{switch} + E(t_{wait})]$$

Here, $E(t_{wait})$ is the energy consumed by waiting for t_{wait} seconds and E_{switch} is the energy consumed by state switches.

The expected energy consumption of waiting for inactivity timer to time out is:

$$\mathbb{E}[E_{no\_switch}] = \int_{t=0}^{t_1+t_2} \mathbb{P}(inter\_arrival\_time = t) \frac{dE(t)}{dt} dt \tag{1}$$

The following expression now is a function of t_{wait}:

$$f(t_{wait}) = \mathbb{E}[E_{no\_switch}] - \mathbb{E}[E_{wait\_switch}]. \tag{2}$$

The best t_{wait} is the one that maximize $f(t_{wait})$, which means that the corresponding value for t_{wait} gives us highest expected gains over the current RRC protocol.

In implementing this algorithm, we take the latest n packets (we discuss how to choose n in Section 6.3) that the control module has seen, to construct the inter-arrival distribution. As new packets are seen, the "window" of the n packet slides forward, and the distribution is adjusted accordingly.

5. MAKEACTIVE ALGORITHM

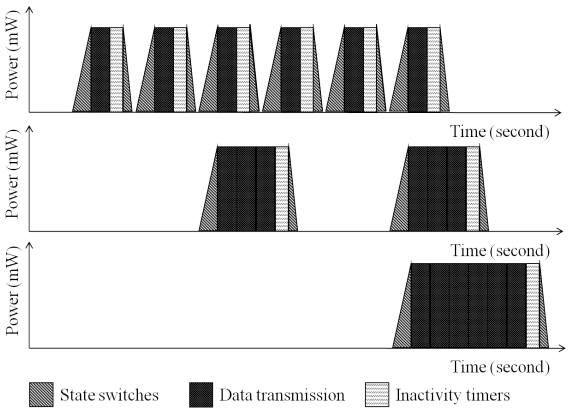


Figure 7: "Shift" traffic to reduce number of state switches.

MakeIdle reduces the 3G wireless energy consumption by switching the radio to Idle mode frequently. Figure 7 (top) shows that MakeIdle may bring more state switches from Idle to Active and from Active to Idle. These switches cause signaling overhead at the base station. One idea to reduce the signaling overhead is to "shift" the traffic bursts in order to combine several traffic bursts together [19, 4], as shown in Figure 7(middle and bottom chart). The longer earlier bursts are delayed, the more bursts we can accumulate and the fewer state switches occur.

In this section, we only consider those background applications for which one can delay the traffic for a few seconds without appreciably degrading the user's experience, not interactive applications where delaying by a few seconds is unacceptable. Our approach differs from previous work [19, 4], where the authors aim to reduce energy consumption by batching bursts of traffic together so that they can share the tail energy. By contrast, because the MakeIdle algorithm already reduces energy by turn radio to the idle mode, MakeActive focuses on reducing the number of state switches to a level comparable to the status quo. As a result, the amount of delay introduced by this method should be much smaller than in previous work.

We first consider a relatively straightforward scheme in which the start of a session (i.e., a burst of packets) can be delayed by at most a certain maximum delay bound, T_{fix_delay}. We then apply a machine learning algorithm, which induces the same number of state switches as the fixed delay bound method, but in addition reduces the delay for each traffic burst. Our contribution lies in the application of this algorithm to learn idle durations for the radio, balancing signaling overhead and increased traffic latency.

5.1 Fixed Delay Bound

A simple strawman is to set a fixed delay bound, T_{fix_delay}. When the radio is in Idle state and a socket tries to start a new session at current time t, and no other such requests are pending, the control module decides to delay turning the radio to Active mode until $t + T_{fix\_delay}$, so that other new sessions that might come between time t and $t + T_{fix\_delay}$ will all get buffered and will start together at time $t + T_{fix\_delay}$. There is a trade-off between the delay bound and the number of sessions that can be buffered. Note that once a session begins, its packets do not get further delayed, which means that TCP dynamics should not be affected by this method.

In the current RRC protocol, the inactivity timers t_1 and t_2 guarantee that after each traffic burst, any new burst comes within $t_1 + t_2$ will not introduce extra state switches between Idle and Active. So in our implementation, we make $T_{fix\_delay} = k \times (t_1 + t_2)$ where k is the average number of bursts during each of the radio's active period.

5.2 Learning Algorithm

The problem with a fixed delay bound is that it does not adapt to the traffic pattern. Every time the delay is triggered, the first transmission may incur a delay of as long as T_{fix_delay}. We show in the evaluation that a large portion of the traffic bursts get delayed by T_{fix_delay}. However, waiting as long as T_{fix_delay} may be overkill; as data accumulates (especially from different sessions), there comes a point when the radio should switch to Active and data sent before this delay elapses, which will reduce the expected session delay while still saving energy.

We apply the *bank of experts* machine learning algorithm [14, 16]. Each "expert" proposes a *fixed* value for the session delay. In each iteration (each time the radio is in Idle mode and a transmission occurs), we computed a weighted average value from the experts and update the weights according to a *loss function*. The process to update each expert's weight is a standard machine learning process, detailed in the appendix.

The loss function is a crucial component of the scheme and depends on the details of the problem to which the learning is applied. Because our goal is to reduce number of state switches by batching, in addition to the delay, the loss function should express the trade-off between the total time delayed for all the buffered sessions and the number of session buffered. The following equation captures this tradeoff:

$$L(i) = \gamma Delay(T_i) + \frac{1}{b}, \gamma > 0$$

Here, γ is a constant scaling parameter between the two parts of the loss function (we chose 0.008 in our implementation because it gave the best energy-saving results among the values we tried). $Delay(T_i)$ is the aggregate time delayed over b sessions, if we choose expert i. b is the number of sessions currently buffered, which is equivalent to the number of state switches avoided. The $1/b$ term ensures that as the number of buffered sessions increases, the value of this part of the loss function reduces, while the other term $\gamma Delay(T_i)$ may increase.

Let t_j be the arrival time of the j^{th} session. Then,

$$Delay(T_i) = \sum_{j=1}^{b} T_i - t_j.$$

6. EVALUATION

We evaluate MakeActive and MakeIdle using trace-driven simulation. We first describe the simulation setup. Then, we evaluate the two methods using traces collected from popular applications run by a few real users. Finally, we compare these methods across different cellular networks.

6.1 Simulation Setup

Energy model. One challenge in our simulations is to accurately estimate the energy consumed given a packet trace containing packet arrival times and packet lengths. Previous work [8] showed that for 3G/LTE, the value of the energy consumed per bit changes as the size of traffic bursts changes. Because our methods may change the size of the traffic bursts, (e.g., MakeIdle may decide to switch the radio to Idle mode within a burst), we build our energy model using the energy consumed per second, which is the power for sending or receiving data.

Network	Sending Power (mW)	Receiving Power (mW)
AT&T 3G	2043	1177
Verizon LTE	2928	1737

Table 1: Average power in mW measured on Galaxy Nexus in Verizon Network. The energy consumed by CPU and screen is subtracted.

Table 1 shows the average power consumed when the phone is sending or receiving bulk data using UDP. Based on this value, we estimate the energy consumed within a traffic burst using the packet inter-arrival time and the packet direction (incoming/outgoing): for each packet reception, the energy consumed is the inter-arrival time multiplied by the average receive power, and similarly for each packet transmission.

To justify this method, we measure the smartphone's energy consumption when it is sending and receiving TCP bulk transfers of different lengths. Each experiment contains five runs. In each run, the phone sends and receives TCP bulk transfers of three lengths (10 kBytes, 100 kBytes and 1000 kBytes) one after another, with a long-enough idle period between each transfer. We find that, on average, the error in the estimated energy consumption is within 10% or less of the true measured value.

One caveat in our energy model is that because fast dormancy is not yet supported on US 3G/LTE networks, we were unable to accurately measure the delay to turn the radio from Active to Idle and the energy consumed. We believe, however, that one can approximate this value by measuring the delay and energy consumed in turning the data connection off on the phone. In practice, we expect the delay and energy of fast dormancy switching to be lower, so we model the turn-off energy and delay for fast dormancy to be 50% of the values measured while turning the radio off. We also evaluated our methods for reasonable fractions (10%, 20%, 40%) other than 50%, and found that the results did not change appreciably; hence, we believe that our conclusions are likely to hold if one were to implement the methods on a device that supports fast dormancy.

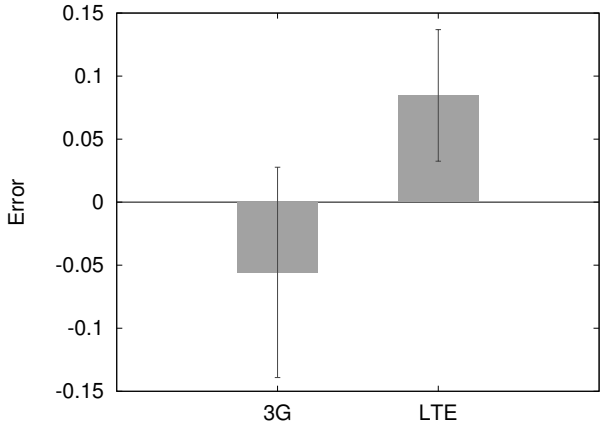


Figure 8: Simulation energy error for Verizon 3G and LTE networks.

Trace data sets. We collected `tcpdump` traces on an HTC G1 phone running Android 2.2 for the seven different categories of applications listed below. For each category, we choose a popular application in the Android Market. Each collected trace was 2 hours long. Most of these applications have the "always on" property in that they usually send or receive data over the network whenever they run, without necessarily requiring user input.

News: A news reader that has a background process running to fetch breaking news.

Instant Message (IM): An IM application that sends heartbeat packets to the server periodically, typically every 5 to 20 seconds.

Micro-blog: A micro-blog application, which automatically fetches new tweets without user input.

Game with ad bar: A game that can run offline, but with an advertisement bar that changes the content roughly once per minute.

Email: This application is run mostly in the background, synchronizing with an email server every five minutes.

Social Network: A user using the social network application to read the news feeds, clicks to see pictures, and posts comments. When running in background, this application updates only every 30 minutes. We did not collect much background traffic from it. We use the foreground traffic trace for comparison trace.

Finance: An application for monitoring the stock market, which updates roughly once per second when running in the foreground.

We also collected real user data from six different users using Nexus S phones in T-Mobile's 3G network and from four different users using Galaxy Nexus phones in Verizon's 3G/LTE network. All the phones run `tcpdump` in the background. Across all users, we collected 28 days of data. For each user, the amount of data collected varies from two to five days.

6.2 Comparison of Energy Savings

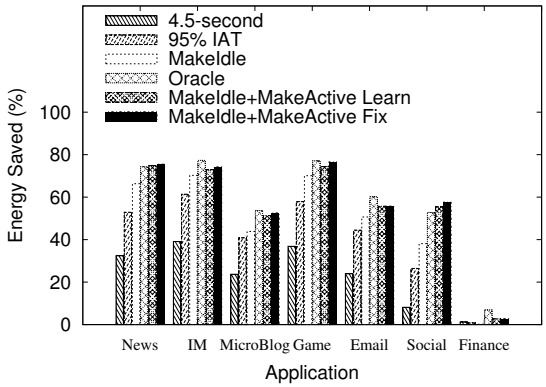


Figure 9: Energy savings for different applications."4.5-second" sets the inactivity timer to 4.5 seconds. "95% IAT" uses the 95th percentile of packet inter-arrival time observed over the entire trace as the inactivity timer. "MakeIdle" shows the energy saved by our MakeIdle algorithm. "MakeIdle +MakeActive Learn" and "MakeIdle +MakeActive Fix" show the energy savings when running MakeIdle together with two different MakeActive algorithms: learning algorithm and fixed delay bound algorithm. Oracle shows the maximum achievable energy savings without delaying any traffic.

We compare MakeIdle against MakeIdle together with MakeActive (shown as MakeIdle+MakeActive), and against two other schemes. The first other scheme is proposed in [6], where a trace analysis found that 95% of the packet inter-arrival time values are smaller than 4.5 seconds. The proposal sets the inactivity timer to a fixed value, $t_1 + t_2 = 4.5$ seconds. We call this approach "4.5-second tail".

The second other scheme is that instead of using the value of 4.5 seconds, we draw the CDF of our traces and get the 95th percentile of packet inter-arrival time observed in each user's trace. We call this approach "95% IAT", which for the data shown in Figure 9 corresponding to one user happened to be 1.67 seconds (the value does vary across users and also across applications). In our evaluation, we are granting this scheme significant leeway because we test the scheme over the same data on which it has been trained. Despite this advantage, we find that this scheme has significant limitations.

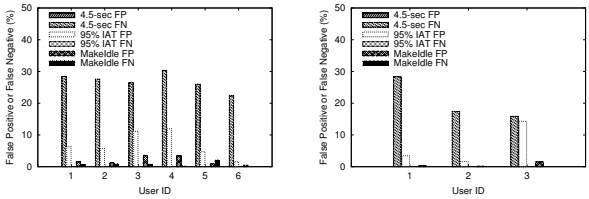
The "Oracle" is an algorithm in which the packet inter-arrival time is known before packet comes, and the algorithm compares the inter-arrival time with the $t_{threshold}$ defined in Section 4.1. The Oracle scheme gives us an upper bound of how much energy can be saved without introducing extra delay. Our MakeIdle + MakeActive algorithm sometimes outperforms the Oracle because it can delay packets and further reduce the number of state switches.

Figure 9 shows that MakeIdle consistently achieves energy savings close to the Oracle scheme, and outperforms the "4.5-second" and "95%" IAT schemes. When both MakeIdle and MakeActive are combined, the savings are greater.

The "95% IAT" scheme gives little or negative savings for "News" and "IM", while the other schemes provide significant positive savings. This is because the 95% percentile of the inter-arrival time is highly variable and cannot guarantee savings in all situations. It is not a robust method.

Figures 10(a) and 11(a) show the estimated energy savings for each user in the Verizon 3G and Verizon LTE networks, respectively. In these results, the different schemes are as explained above, except that the 95% IAT scheme uses per-user (but not per-application) inter-arrival time CDFs. The gains of MakeIdle and MakeActive over the other schemes are substantial in most cases. In the LTE case, the 95% IAT scheme sometimes saves the most energy (for user 2 and user 3), but sometimes performs worse than MakeIdle (for user 1); it depends on the user, again showing a lack of robustness. Perhaps more importantly, the number of state switches is enormous compared to the other schemes, making it extremely unlikely to be useful in practice.

6.3 MakeIdle Evaluation



(a) Verizon 3G. (b) Verizon LTE.

Figure 12: False ("FP" short for false positive) and missed switches ("FN" short for false negative).

To understand why MakeIdle outperforms the other methods, we calculate the fraction of *false switches* and *missed switches* for each method. We use "Oracle" as ground truth and define these ratios as follows:

$FalseSwitch(FalsePositive) = N_{FS}/(N_{FS}+N_{TN})$. Here, N_{FS} is the number of cases where the algorithm switches the radio to Idle but Oracle decides to keep the radio in Active mode. N_{TN} is the number of cases where both Oracle and the algorithm decide to keep the radio Active.

$MissedSwitch(FalseNegative) = N_{MS}/(N_{MS}+N_{TP})$. Here, N_{MS} is the number of cases where the algorithm decides to keep the radio in the Active mode but Oracle switches the radio to Idle. N_{TP} is the number of cases where both Oracle and the algorithm switch the radio to Idle. A high missed switch value means the algorithm tends to keep the radio in Active mode, which may not be energy-efficient.

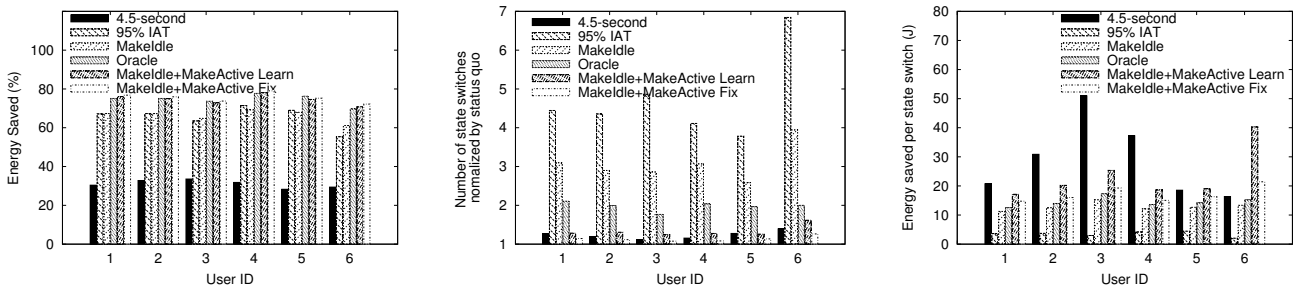
Figure 12 shows these two ratios for different data sets. Note that these values for MakeIdle are much smaller than for the other two algorithms.

Figure 13 shows the false positive and false negative rates (in percentage) as a function of the number of recent packets used to construct the distribution defined in Section 4.2. We find that the false negative rate is relatively constant, while the false positive rate decreases as the window size increases. For all the other results shown in §6, we use $n = 100$.

Another factor that affects battery consumption is the waiting time between a packet arrival and the time at which the algorithm actually switches the radio to Idle. For the "4.5-second tail" scheme, the waiting time is always 4.5 seconds. Similarly, the waiting time for "95% IAT" is 0.85 seconds for 3G and 0.01 seconds for LTE. In contrast, MakeIdle chooses the waiting time dynamically, achieving better gains. Figure 14 shows an example of waiting time changes in a user's trace in Verizon 3G network.

6.4 MakeActive Evaluation

Although shortening t_{wait} with the MakeIdle algorithm saves considerable amounts of energy, it may bring about more state switches between the Low-power idle and Active states. But when there are multiple applications running at the same time, or when one

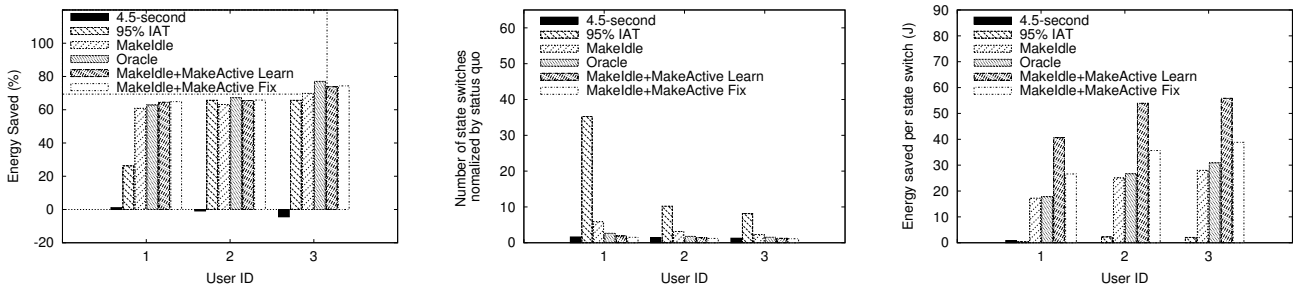


(a) Energy savings.

(b) Number of state switches normalized by status quo.

(c) Energy saved per state switch.

Figure 10: Energy savings and signaling overhead (number of state switches) across users in the Verizon 3G network.



(a) Energy savings.

(b) Number of state switches normalized by status quo.

(c) Energy saved per state switch.

Figure 11: Energy savings and signaling overhead (number of state switches) across users in the Verizon LTE network.

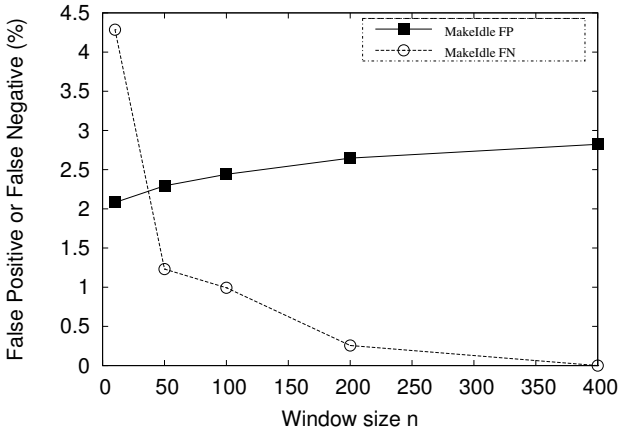


Figure 13: False ("FP") and missed switches ("FN") changes as the number of packets used to construct distribution defined in Section 4.2.

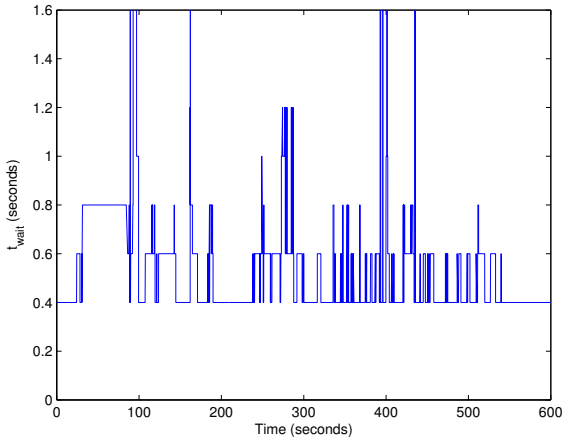
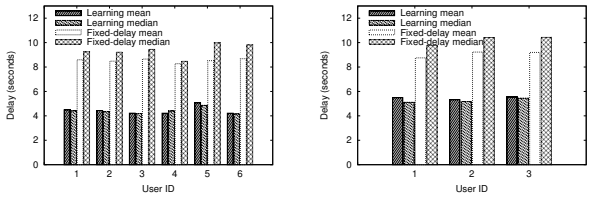


Figure 14: Waiting time changes in MakeIdle.

application starts multiple connections, we can reduce the number of state switches by delaying the connections and batching them together using MakeActive.

Figures 10(b) and 11(b) show the number of state switches using different algorithms, normalized by the number measured in the status quo. Each user has several applications running on the phone. For MakeIdle only, in the 3G/LTE network, the number of state switches is at most four to five times higher than the status quo. For MakeIdle with MakeActive, either using the learning algorithm or the fixed-delay bound, the number of state switches is about the same as the status quo, meaning that by delaying traffic bursts, our algorithm can reduce the energy consumption without introducing any extra signaling overhead. Notice that for the "95% IAT" algorithm in the LTE network, the number of state switches is as high as 35× the status quo because the corresponding timer value is only 0.01 seconds. As a result, this method will always switch the radio



(a) Verizon 3G network. (b) Verizon LTE network.

Figure 15: Mean and median delays for traffic bursts using learning algorithm and fixed delay bound scheme.

to Idle even if there is only a small gap between packets. In a few cases, that does save energy, but at great expense.

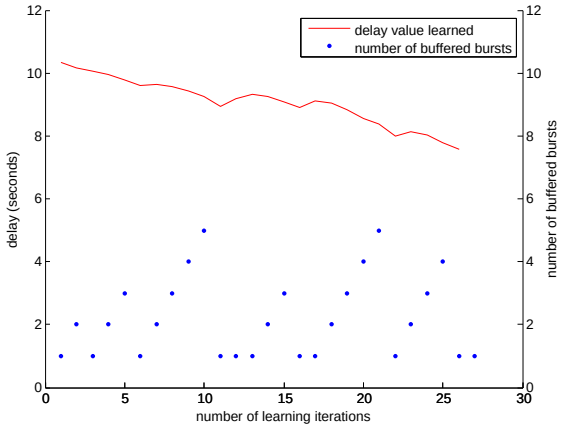


Figure 16: Delay value changes as the learning proceeds.

In Section 5, we described both the fixed-delay bound and a learning algorithm. Figure 15 shows that using the learning algorithm reduces the average delay for each traffic burst by 50% compared to the fixed-delay bound, while both methods induce a comparable number of state switches (Figure 10(b) and Figure 11(b)). The learning algorithm is able to reduce the delay because the loss function (defined in Section 5.2) balances the tradeoff between the number of buffered bursts and the total delay. Figure 16 shows that due to the loss function, the algorithm will reduce the delay bound as the number of buffered bursts increase.

6.5 Different Carriers

To gain a better understanding on how different carriers' RRC state machine configurations affect the observed improvement, in this part of the evaluation we run our trace-driven simulation on different RRC profiles measured from the four major US carriers. In Table 2 we list the measured RRC parameters. There are two cases where the inactivity timer $t_2 = 0$ (effectively), because we cannot clearly distinguish t_1 and t_2 from the energy difference.

Figure 17 shows the percentage of energy saved compared to the status quo. Figure 18 shows the corresponding signaling overhead. We find that the "MakeIdle+MakeActive" method outperforms the "4.5-second tail" method in all the carrier settings. Figure 18 shows the number of state switches (proportional to signaling overhead) of different schemes divided by the number of state switches without using any scheme.

The maximum signaling overhead for MakeIdle is less than 3.1× the baseline where no fast dormancy is triggered. For "MakeIdle+MakeActive", the signaling overhead reduces to only 1.33× or less, a 62% reduction from the previous 3.1×, and is close to the signaling overhead of "4.5-second tail". The session delays brought by MakeActive are listed in Table 3.

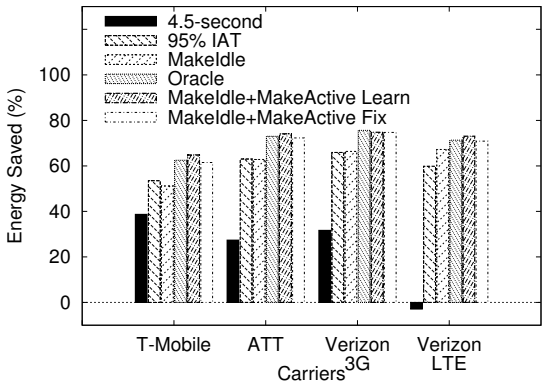


Figure 17: Energy saved for different carrier parameters using different methods. For "MakeIdle", the maximum gain is 67% in Verizon LTE netwrok. For "MakeIdle+MakeActive", the maximum gain is 75% achieved in Verizon 3G.

In both Figure 17 and 18, the result shown as MakeIdle has no traffic batching, which corresponds to the case when all the traffic is treated as delay-sensitive, for example, web browsing. The MakeActive method is disabled in this case to make sure that the user's experience is not adversely affected. One possible method to decide when to disable MakeActive is for the control module maintain a list of delay-sensitive or interactive applications; when any of these applications is running in the foreground, the system disables MakeActive.

Even without MakeActive, the reduction in energy consumption is still significant in all the 4 carrier settings. The maximum gain is for Verizon LTE, where MakeIdle save 67% energy over status quo. With MakeIdle, the maximum gain is Verizon 3G, where the energy saving reaches 75%, and the corresponding median delay is 4.48 seconds.

6.6 Energy overhead of running algorithms

To measure the Energy overhead of running our methods, we

Network	P_{snd}	P_{rcv}	P_{t_1}	P_{t_2}	t_1	t_2
T-Mobile 3G	1202	737	445	343	3.2	16.3
AT&T HSPA+	1539	1212	916	659	6.2	10.4
Verizon 3G	2043	1177	1130	1130	9.8	0
Verizon LTE	2928	1737	1325	-	10.2	-

Table 2: Power and inactivity timer values for different networks. Power values are in mW, times are in seconds.

Network	Mean Delay	Median Delay
T-Mobile 3G	5.11	5.11
AT&T HSPA+	4.80	4.65
Verizon 3G	4.67	4.48
Verizon LTE	4.62	4.38

Table 3: The mean and median session delays brought by MakeIdle for different carriers (in seconds).

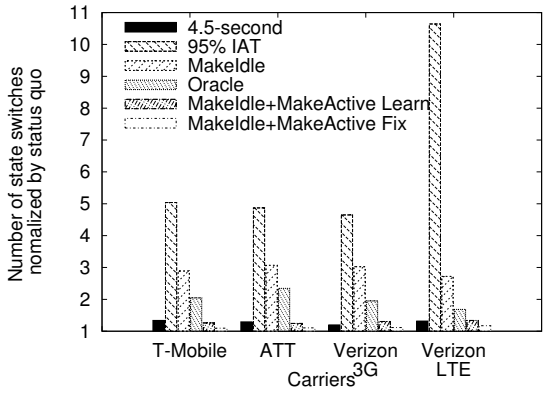


Figure 18: Number of state switches (signaling overhead) for different methods divided by number of state switches using the current inactivity timers.

implemented the algorithms on our test phones. We then generated traffic from the phone based on the user traces we collected. We ran the traffic generator with and without our methods enabled, ensuring that it generates the same traffic in all the experiments. We used the power monitor to measure the total energy consumed in both cases. The energy overhead for running our algorithm is 1.7% for AT&T HTC Vivid and 1.9% for Verizon Galaxy Nexus.

7. RELATED WORK

We divide related work into measurement studies of 3G energy consumption and approaches to reduce that energy, 3G usage profiling, and WiFi power saving methods.

3G energy mitigation strategies:

Past work aimed at eliminating the tail energy falls into three categories: inactivity timer reconfiguration, tail cutting, and tail sharing.

Inactivity timer reconfiguration. Lee et al. [11] developed analytic models for energy consumption in WCDMA and CDMA2000 and showed that the inactivity timer should be dynamically configured. Falaki et al. [6] proposed an empirical method by plotting the CDF of packet inter-arrival times for traces collected on smartphones communicating over 3G radio over long period of time (several days). They found that 95% of the packet inter-arrival time values are smaller than 4.5 seconds, and proposed setting the inactivity timer to a fixed value, $t_1 + t_2 = 4.5$ seconds. Our approach finds a dynamic inactivity timer value using traffic pattern information within a short period of time.

Tail cutting. Qian et al. [19] gave an algorithm, *TOP*, to help the device decide when to trigger fast dormancy based on the information provided by applications running on the device. Their algorithm requires the application to predict when the next packet will come and report it to the OS. This approach requires modifications to the applications, and it is not clear how each application should make these predictions. Our work requires no modification to the application code and does not require the application to predict its traffic.

Traffic batching. Balasubramanian et al. [4] propose an application-layer protocol, *TailEnder*, to coalesce separate data transfers by delaying some of them. For delay-tolerant applications such as email, TailEnder allows applications to set a deadline for the incoming transfer requests; they suggest and evaluate a relatively long delay of 10 minutes for such applications. For applications that can benefit from prefetching, TailEnder prefetches 10 web documents for each user query. Their design need to re-implement the application and let each application propose their own delay tolerant timers, whereas our design is able to "pause" the traffic transmission at OS layer.

Liu et al. [12] proposed *TailTheft*, a traffic queuing and scheduling mechanism to batch traffic among different applications and share the tail energy among them. One idea of this work is to setup a timeout value for delay-tolerant transfers, and transfer data when timeouts or other delay-sensitive transfer have triggered the radio to Active mode. Similar to *TailEnder*, they require the application to specify how much delay is acceptable.

Another traffic batching approach is prefetching. Qian et al. [18] proposed a prefetching algorithm for YouTube, which erases the tail between transfers of video pieces.

3G resource usage profiling:

Qian et al. [17] designed an algorithm to infer RRC state machine states using packet traces. The per-application analysis shows that some of the popular mobile applications have traffic patterns that are not energy-efficient, due to low bit-rate transmission, inefficient prefetching, and aggressive refresh.

WiFi power-saving algorithms:

Much prior work has focused on WiFi power-saving algorithms [9, 10, 20]. The problem in WiFi networks is qualitatively different from 3G; in WiFi, the time and energy consumed to transition between states is negligible; what is important is to dynamically determine the best sleep duration when the WiFi radio is off. In this state, no packets can be delivered, but the access point will be able to buffer them; the problem is finding the longest sleep time that ensures that no packets are delayed (say, by a specified maximum delay). In the 3G context, changing the state of the radio consumes time, energy, and network signaling overhead, but there is no risk of receiving packets with excessive delay because the base station is able to notify a mobile device that packets are waiting for it even if the device is in Idle state. Thus, we cannot simply apply WiFi power-saving algorithms to 3G networks. Also, machine learning algorithms has been applied to the 802.11 power saving mode configuration problem [15], but the problem setup is different for the 3G energy environment because of different tradeoffs we aim to balance.

Power-saving for processors:

Though not directly related to the problem we address, previous work on processor power-saving has used a similar model to us in which the different power states and transitions between different states are abstracted as a state machine [5]. Here, the power-saving mechanisms are categorized into static methods and adaptive methods, with the adaptive methods using a nonlinear regression over previous idle/active periods and knowledge of how successful previous power-saving decisions are.

8. CONCLUSION AND FUTURE WORK

3G/LTE energy consumption is widely recognized to be a significant problem [4]. We developed a system to reduce the energy consumption using knowledge of the network workload. In evaluating the methods on real usage data from 9 users over 28 total days on four different carriers, we find that the energy savings range between 51% and 66% across the carriers for 3G, and is 67% on the Verizon LTE network. When allowing for delays of a few seconds (acceptable for background applications), the energy savings increase to between 62% and 75% for 3G, and 71% for LTE. The increased delays reduce the number of state switches to be the same as in current networks with existing inactivity timers.

The key idea in this paper is to adapt the state of the radio to network traffic. To put the 66% saving (without any delays) or 75% saving (with delay) in perspective, we note that according to the Nexus S specifications, the reduction in lifetime from using the 3G radio instead of 2G is 7.3 hours; while it is not clear what application mix produces these numbers, one might speculate that saving 66% of the energy might correspond to an increase in lifetime by about 66% of 7.3 hours, or about 4.8 hours.

There are two areas for future work. First, studying the effects of triggering fast dormancy on the base station side would be useful, considering issues such as handling multiple phones triggering the feature, and whether the base station can actively help the phone to make decisions on fast dormancy by buffering incoming traffic for the phone. Second, extending the system to include server or base station functions to coordinate with the mobile device to further reduce energy consumption.

9. ACKNOWLEDGMENTS

We thank Katrina LaCurts, Lenin Ravindranath, Keith Winstein, Jonathan Perry, and Raluca Ada Popa for useful comments on earlier versions of this paper. We also thank the reviewers and our shepherd, Srikanth Krishnamurthy, for their insightful comments. This material is based upon work supported by the National Science Foundation under Grant No. CNS-0931550.

10. REFERENCES

[1] 3GPP Release 7: UE Fast Dormancy behavior, 2007. 3GPP discussion and decision notes R2-075251.

[2] 3GPP Release 8: 3GPP TS 25.331, 2008.

[3] Data Service Options for Spread Spread Spectrum Systems: Service Options 33 and 66, May 2006.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Internet Measurement Conference*, 2009.

[5] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299 –316, june 2000.

[6] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. In *Internet Measurement Conference*, 2010.

[7] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *MobiSys*, 2012.

[9] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *MobiCom*, 2002.

[10] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. *ACM Wireless Networks*, 11(1–2):135–148, Jan. 2005.

[11] C.-C. Lee, J.-H. Yeh, and J.-C. Chen. Impact of inactivity timer on energy consumption in WCDMA and CDMA2000. In *IEEE Wireless Telecomm. Symp.(WTS)*, 2004.

[12] H. Liu, Y. Zhang, and Y. Zhou. TailTheft: Leveraging the wasted time for saving energy in cellular communications. In *MobiArch*, 2011.

[13] Monsoon power monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[14] C. Monteleoni. Online learning of non-stationary sequences. In *AI Technical Report 2003-011, S.M. Thesis*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 2003.

[15] C. Monteleoni, H. Balakrishnan, N. Feamster, and T. Jaakkola. Managing the 802.11 energy/performance tradeoff with machine learning. Technical Report MIT-LCS-TR-971, MIT CSAIL, 2004.

[16] C. Monteleoni and T. Jaakkola. Online learning of non-stationary sequences. In *Neural Information Processing Systems 16*, Vancouver, Canada, December 2003.

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

[18] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3G networks. In *Internet Measurement Conference*, 2010.

[19] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. TOP: Tail Optimization Protocol For Celluar Radio Resource Allocation. In *ICNP*, 2010.

[20] T. Simunic, L. Benini, P. W. Glynn, and G. D. Micheli. Dynamic power management for portable systems. In *MobiCom*, 2000.

[21] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. Comparative Analysis of Energy-Saving Techniques in 3GPP and 3GPP2 Systems. *IEEE Trans. on Vehicular Technology*, 58(1):432 –448, Jan. 2009.

[22] J.-H. Yeh, C.-C. Lee, and J.-C. Chen. Performance analysis of energy consumption in 3GPP networks. In *IEEE Wireless Telecomm. Symp. (WTS)*, 2004.

APPENDIX

Here we show how *bank of experts* works. We bound the maximum delay to n seconds. Each expert "proposes" a delay value T_i:

$$T_i = i,\ i \in 1 \ldots n.$$

The output of the algorithm is the weighted average over all the experts:

$$T_t = \sum_{i=1}^{n} p_t(i) T_i$$

For each iteration of the updates, the algorithm calculates the probability of each possible hidden state (in our case, the identity of the expert) based on some observation y_t. Here, we can define the probability of predicting observation y_t as $P(y_t|T_i) = e^{-L(i,t)}$. The observation is the number of sessions we batched at time t, and $L(i,t)$ is the loss function. Then we can apply the following equation to get the weight $p_t(i)$:

$$p_t(i) = \frac{1}{Z_t} \sum_{j=1}^{n} p_{t-1}(j)\, e^{-L(j,t-1)}\, P(i|j,\alpha).$$

Here, Z_t is a normalization factor that makes sure $\sum_i p_t i = 1$. The $P(i|j,\alpha)$ shows the probability of switching between experts. There are different versions to solve this part. The one we chose [7] supports switching between the experts and is suitable for cases where the observation may change rapidly, which matches the bursty character of network traffic. $P(i|j,\alpha)$ is defined as:

$$P(i|j,\alpha) = \begin{cases} (1-\alpha) & i = j \\ \frac{\alpha}{n-1} & i \neq j \end{cases}$$

$0 \leq \alpha \leq 1$ is a parameter that determines how quickly the algorithm changes the best experts. α close to 1 means the network condition changes rapidly and the best expert always changes. One

problem with this algorithm is that it is hard to choose a good α. In reality, α should not be a fixed value since the network traffic pattern may change rapidly or remain stationary. We use a more adaptive algorithm, Learn-α [14, 16], to dynamically choose α.

The basic idea is to first assign m α-experts and use the algorithm above to learn the proper value of α in each iteration, and then use the up-to-date α to learn T_t [14, 16]. The final equation for this "two-layer learning" is:

$$T_t = \sum_{j=1}^{m} \sum_{i=1}^{n} p'_t(j) p_{t,j}(i) T_i \tag{3}$$

Here, $p'_t(j)$ is the weight for the j^{th} α-expert, which is given by:

$$p'_t(j) = \frac{1}{Z_t} p'_{t-1}(j) e^{-L(\alpha_j, t-1)} \tag{4}$$

This equation shows that $p'_t(j)$ is updated from the previous value $p'_{t-1}(j)$; the initial values are: $p'_1(j) = 1/m$. $-L(\alpha_j, t-1)$ is the α loss function, defined as:

$$L(\alpha_j, t) = -\log \sum_{i=1}^{n} p_{t,j}(i) e^{-L(i,t)} \tag{5}$$

Here, $L(i,t)$ is the loss function, discussed in §5.2. t is the present time; the loss function value for the current iteration is calculated from information learned at time $t-1$.