Energy-aware Cross-layer Burst Buffering for Wireless Communication

Ekhiotz Jon Vergara, Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University, Sweden
{ekhiotz.vergara,simin.nadjm-tehrani}@liu.se

ABSTRACT

The massive explosion of mobile applications with the ensuing data exchange over the cellular infrastructure is not only a blessing to the mobile user but also has a price in terms of regular discharging of the device battery. A big contributor to this energy consumption is the power hungry wireless network interface. We leverage a measurement kit to perform accurate physical energy consumption measurements in a third generation (3G) telecommunication modem thus isolating the energy footprint of data transfers as opposed to other mobile phone-based measurement studies. Using the measurement kit we show how the statically configured network parameters, i.e., channel switch timers, and buffer thresholds, in addition to the transfer data pattern and the radio coverage, impact the communication energy footprint. We then demonstrate that being aware of static network parameters creates room for energy savings. This is done by devising a set of algorithms that (a) infer the network parameters efficiently, and (b) use the parameters in a new packet scheduler in the device. The combined regime is shown to transfer background uplink data, from real world traces of Facebook and Skype, with significant energy saving compared to the state-of-the-art.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Wireless communication; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Algorithms, Design, Measurement

Keywords

energy consumption; UMTS; mobile devices; scheduling

1. INTRODUCTION

Mobile computing is on the verge of entering a new era in which the vision of "information anytime anywhere" is becoming a reality. Unfortunately, battery technology has not kept up with this evolution making the new power hungry capabilities also a hinder for further development. Ubiquitous connectivity and current mobile data plans have led to a significant increase in the use of the radio hardware interfaces, with the outcome that the battery of a modern smartphone hardly lasts more than one day.

While the technology development may provide more energy-efficient hardware and batteries in the longer run, we believe there is still a need for carefully analyzing the energy footprint of application software, and to use software to monitor and reduce the energy consumption. However, in the device-user-network power equation the user application impacts and potentials are often neglected. While a lot of effort is directed towards lowering energy consumption in the infrastructure nodes, and competition drives a reduction on base energy in handheld devices, the user application contributions to the power equation are less studied. Most applications are completely oblivious to how their data affects or is affected by lower layer interactions.

Our work starts by illustrating that the third generation (3G) wireless communication power footprint at the user end is not only affected by the amount of data transferred, but also affected by receive signal strength and the static network parameters configured at the network end. The Radio Resource Control (RRC) protocol drastically influences the energy per bit in a transmission, whereby a few small text messages may consume as much as a videoconference at the user end. By performing measurements on transmission energy at the device end we show that "hidden" parameters of the network like inactivity timers for switching between channels, or data buffer thresholds can be used to schedule the transmission of packets in an energy-efficient manner. But before doing that we propose methods for inferring these parameters. The contributions of our work are as follows:

- The impact of radio layer statically configured parameters on the energy consumption is physically measured in a broadband module. The study includes the inactivity timers and the data buffer thresholds used by the operator for state transition decisions as well as the radio coverage.
- Algorithms to estimate the inactivity timers and buffer thresholds which are at least 65% more efficient in terms of energy consumption and twice faster compared to the state-of-the-art.
- An algorithm that schedules background data transmissions based on the aforementioned network param-

e-Energy 2012, May 9-11 2012, Madrid, Spain.

eters saving up to 35% of energy for common applications like Facebook and Skype.

The rest of our work is organised as follows: section 2 describes the basic background about the third generation Universal Mobile Telecommunication System (UMTS) and introduces the related work. Section 3 presents our measurement physical setup and describes the performed energy consumption measurements. Section 4 describes the algorithms to estimate inactivity timers and buffer thresholds and presents our cross-layer scheduling algorithm. Section 5 presents the evaluation and results of our approach and section 6 concludes the paper.

2. BACKGROUND AND RELATED WORKS

This section introduces the necessary background for following the results of our paper and presents the main works related to our study.

2.1 3G background

The Radio Network Controller (RNC) is a key element in the UMTS Terrestrial Radio Access Network (UTRAN). It is responsible for the radio resource management and also manages the Node Bs (also known as base transceiver station), to which the user equipment (UE) connects via radio physical channel. The energy consumption of the UE in 3G is mostly influenced by the RRC and the Radio Link Control (RLC) protocols, which are defined in the UMTS Wideband Code Division Multiple Access protocol stack.

According to RRC the UE can be in the states depicted in Fig. 1. The states are placed along the y and x axis according to their power consumption and performance in terms of response time and maximum data rate respectively.

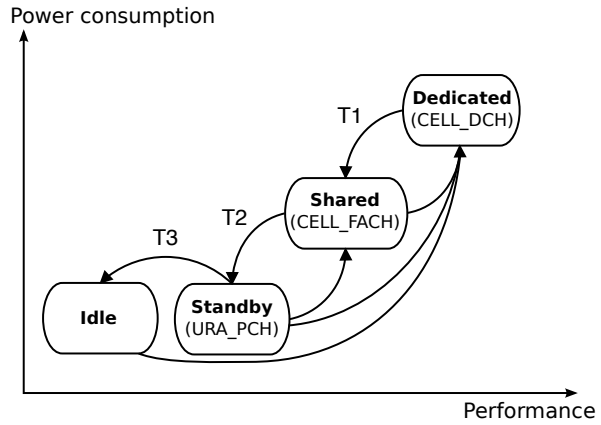

Figure 1: 3G connection states

RRC States: in the Dedicated state, a dedicated physical channel (CELL_DCH) is allocated for the terminal in both uplink and downlink providing higher data rates. The terminal has access to dedicated uplink or downlink transport channels, shared transport channels and a combination of them. In the Forward Access Channel (CELL_FACH) the terminal is assigned a default common or shared transport channel in the uplink and monitors the downlink continuously. The UE can transmit small data packets at lower data rates. While in CELL_DCH and CELL_FACH the UE remains connected to the RNC.

The UE is in the Idle state when there is no network activity. It is not connected but it still can check if there is any downlink packet available. Denoted as Standby in Fig. 1, the 3G standard also describes two optional states where the UE maintains a connection to the RNC and the energy consumption is similar to Idle state: the Paging Channel (CELL_PCH) and UTRAN Registration Area Paging Channel (URA_PCH). These two states allow the UE to switch faster to higher states. Note that some operators do not implement the optional states. In our work, the UE was connected to an operator that implements the URA_PCH state. For the rest of the document we will refer to the different states as DCH, FACH, PCH and Idle.

State transitions: state transitions on the UE occur based on traffic volume and inactivity timers controlled by the RNC. Statically set inactivity timers control the state transitions DCH-FACH, FACH-PCH and PCH-Idle, T1, T2 and T3 in Fig. 1 respectively. E.g., when the UE is in the DCH state for T1 without any or small data transmission, the RNC releases the dedicated channel and switches the UE to FACH by means of the RRC protocol.

The RRC uses information from the RLC protocol [14] in order to report the observed traffic volume to the network. For example, in FACH, the UE reports to the RNC the observed traffic volume based on data buffer status. This helps the RNC to re-evaluate the allocation of resources. The RLC data buffer is used to trigger state transitions, which can be downlink or uplink. When the content of the data buffer exceeds a certain threshold, the corresponding signaling is performed before switching the state. These buffers are cleared out when the data is transmitted. In section 4.3 we will refer to T_{clear} denoting the duration after which the buffers are cleared after transmitting the data, which depends on the data rate of the allocated channel.

Network coverage: the UE is exposed to different network coverages and therefore to different received signal strengths (RSS) [20]. The UE uses link adaptation adjusting data rate to compensate for the different channel conditions that the user equipment is exposed to. The radio power amplifier of the UE also increases its gain to compensate for the drop in RSS and reduce communication errors. When the radio link quality is low, the UE consumes more energy due to higher power consumption and lower data rate.

2.2 Related works

In recent years there has been a growing interest in studying energy consumption in mobile devices and green networking. The main body of work related to our study can be categorised into two areas: energy consumption studies of wireless networking and energy-saving techniques. Some proposals consider only infrastructure-centric approaches [5, 10,11], whereas our focus is on reducing the energy consumption on the UE side.

Energy consumption studies: experimental setups for energy consumption studies on mobile devices either use built-in software in mobile devices like Nokia Energy Profiler (NEP) [3, 12, 15, 22] or external power meters [9, 18–20, 23]. The main advantage of our measurement setup is that we are able to isolate the energy consumption of data transfers by performing measurements in a 2G/3G/GPS module.

An influential measurement study using NEP by Balasubramanian et al. [3] reports the categorisation of three different energy components in cellular networks: ramp, transfer and tail. The tail component is the most significant and is caused by the inactivity timer statically set by the net-

work operators. Our previous work [2] refines this study by performing physical measurements using a cellular modem isolating the energy of data transfers. This helps to isolate the impact of system software and unsolicited network connections.

A detailed study of the tail energy overhead in 2G/3G cellular networks is performed by Qian et al. [18] on user data traces retrieved from a network operator in 2009. The work is extended [17] pointing out how different applications inefficiently utilise the radio resources due to their data pattern. Both works provide an excellent ground for understanding the impact of the operator-configured timers and RLC data buffer thresholds on the theoretical user battery discharge.

Perrucci et al. [13] focus their study on the energy consumption difference of 2G and 3G according to common usage of the networks, such as SMS, voice calls and data traffic. Wang et al. [24] present the energy consumption of data transfers over WLAN and cellular networks. Energy per bit and energy per second, i.e. average power drawn in a second, are the metrics used for comparison. Puustinen et al. [15] measure the impact of unwanted Internet traffic on the energy consumption. An application-centric energy study of mobile Youtube is performed by Xiao et al. [25] showing that the download of videos in WLAN is more energy efficient than in 3G. Schulman et al. [20] explore the impact of the network coverage on the energy consumption.

The above mentioned works consider a combined energy consumption in the mobile device (CPU, memory access and transmission dependent energy use). The main advantage of our study is the fine-grained measurements isolating data transfer energy and extension of previous studies identifying the detailed effects of the poor radio links in the energy consumption.

Energy-saving techniques: the work closest to ours is TailEnder by Balasubramanian et al. [3]. The concept of Delay Tolerant Applications (DTA) is introduced via the intuition that to reduce energy consumption some applications can defer their transmissions based on users demands. TailEnder uses an online scheduling algorithm with user specified deadlines leading to traffic aggregation and bursty transmissions while reducing the number of state transitions and energy. We use the results of this work as our base line. Several similar approaches are found in the literature inspired by TailEnder: Könönen et al. [6] propose the alignment of timers of different applications in order to perform synchronised bursty transmissions. The proposal of Calder et al. [4] schedules data transmissions of applications with different transmission intervals in multiples of the shortest interval. TailTheft [8] schedules bursty data transfers of DTA in a similar way to TailEnder and uses the Fast Dormancy (FD) of the 3GPP Release 8 standard, which allows the UE to signal the RNC in order to release the connection and go to Idle or Standby earlier.

Using FD, Puustinen et al. [15] reduce the energy consumption of unwanted Internet traffic. Qian et al. [16] assume that applications know their inter-packet time and can signal the RNC to release the connection by means of FD. Bartendr [20] exploits the fact that transmitting is less costly when the received signal is strong by transmitting data in periods of good signal strength in cellular networks.

In comparison, our work is the first in considering the RLC data buffer thresholds to schedule small data transmissions of DTA in FACH in order to minimise the use of the more energy consuming DCH state. Approaches using FD are complementary to our work which would allow us to release the RNC connection when needed.

The concept of traffic backfilling [7] is proposed to opportunistically transmit data during unused gaps between interactive traffic. Our previous work [2] introduces the concept of burst buffering for live streaming that buffers a segment of the data and then sends it in burst. The intuition is that the energy cost is more or less the same no matter the constant data rate. Armstrong et. al [1] present an energy-efficient Web browsing content update combining the reduction of transferred data by using caches and Web page updates via bursty data transmissions from a proxy. Cool-Tether [21] is a WiFi hotspot focused on serving web pages using the cellular connection of various smartphones.

In summary, our work is the first in considering both the tail energy overheads and the RLC data buffers in combination with scheduling data transfers. We infer the tails at least 2 times faster using 65% less energy than previous works [18] and use the knowledge of the radio layer in order to schedule DTA data transmissions more efficiently in terms of energy.

3. ENERGY MEASUREMENT STUDY

In this section we begin by an experimental study of the main factors affecting the energy consumption of 3G for the UE. All the energy measurements were performed using the physical setup that we describe below, allowing the isolation of energy consumption of data transfers.

3.1 Physical setup

All the measurements were performed on a power-efficient mobile broadband module (Ericsson F3307) which provides 2G, 3G and GPS connectivity. It is designed to provide mobile broadband to consumer electronic devices such as tablets and laptops. The module provides uplink and downlink speeds up to 5.76 and 7.2 Mbps respectively implementing the High Speed Packet Access (HSPA) standard.

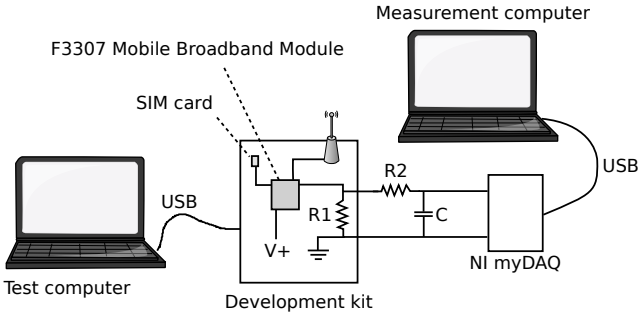

Figure 2: Measurement setup

The measurement setup depicted in Fig. 2 consists of the mobile broadband module placed on an Ericcson's Developer Starter Kit. The kit provides network connectivity via USB to a test computer that runs Ubuntu 10.10 with the firewall activated in order to allow only the desired network connections. The measurements were performed using a National Instruments myDAQ data acquisition device sampling the voltage drop over a shunt resistor of 0.1 (R1 in Fig. 2) at 1 kHz. The power consumption is derived from the voltage and in order to avoid any anti-aliasing effects a low-pass filter of approximately 16 Hz (R2 = 10 k and C = 1 uF in the

figure) was added.

All measurements were performed using a SIM card from TeliaSonera providing full access to the available capacity of the Sweden 3G network[1]. Unless specified, all the measurements were performed in the same location at university where the received signal strength did not vary significantly.

3.2 3G energy consumption

The energy footprint of 3G is mostly influenced by the state machine described in Section 2.1.

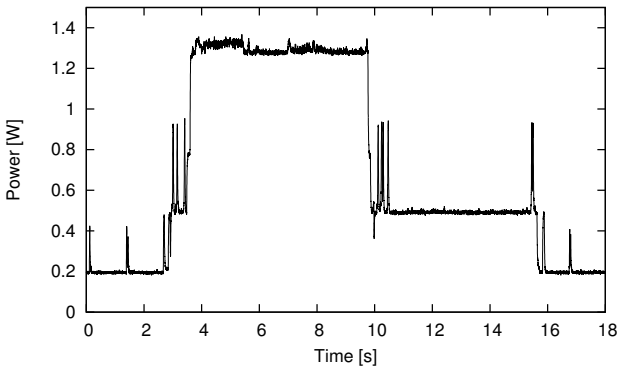

Figure 3: Power consumption of different 3G states

Fig. 3 shows the power consumption levels of the states implemented by TeliaSonera when downloading data. The 3G module stays in PCH state until there is a transition to DCH, after the dedicated channel connection has been stablished (3s in Fig. 3). Once the UE is on DCH the data is downloaded. The transition to FACH occurs when all dedicated channels have been released (10s in Fig. 3). The power consumption in the DCH state is around 1.3 Watts, higher than the FACH state (around 0.5 Watts) and PCH state (0.2 Watts).

It can be noticed that after downloading the data, the UE continues in the DCH state until the inactivity timer T1 has expired causing an energy consumption overhead. A similar overhead is caused in FACH due to T2. The overheads caused by the inactivity timers are called tail effects [3]: T1 leads to DCH_TAIL and T2 to FACH_TAIL. Our previous work [2] studied the energy overhead caused by the inactivity timers. Since the power consumption of PCH and Idle is similar, we do not consider the effect of T3.

The uplink and downlink RLC data buffers are used to estimate the traffic volume and trigger state transitions to higher states. When the data in buffer exceeds the threshold, the RNC re-evaluates the allocation of resources and up-switches the state of the UE. We measured the uplink and downlink thresholds that trigger the state transitions shown in Table 1, by sending UDP packets of different sizes and observing the live power trace using our measurement setup. Note that the PCH-DCH transition is more probable to happen when the triggering packet is closer to the upper bound, i.e., transmitting 513 bytes uplink might not lead to a state transition.

Our measurements were carried out over an interval of 5 months, and 3 months into the period the operator settings were changed. The value of the uplink RLC data buffer threshold that triggers PCH-DCH transition was incremented from the value of Table 1 to around 900 bytes (875 - 1000 bytes).

[1]TeliaSonera is currently the 5th largest operator in Europe with around 157 million customers.

Table 1: State transitions and triggering packet sizes.

State Transition	Uplink Size (bytes)	Downlink Size (bytes)
PCH-DCH	513 - 542	524 - 558
FACH-DCH	294	515
PCH-FACH	Always triggered	

The tail timers and RLC buffer thresholds vary per operator. The variation and the data traffic pattern highly influences energy consumption of the UE. Sporadic transmissions of small packets can lead to high energy consumption. As an illustrative example, we sent a 600 bytes UDP packet every 3 seconds during 1 minute which led to a similar consumption to a Skype voice call (average of 19kB/s): 90 and 92 Joules respectively. The knowledge of tail overheads and RLC buffer thresholds can benefit the UE which would be able to predict or "control" state transitions in order to reduce energy consumption.

The energy consumption of the UE also varies significantly with the radio coverage. A clear example of this phenomenon is depicted in Fig. 4, where a 4MB file download is performed at different locations at different RSS.

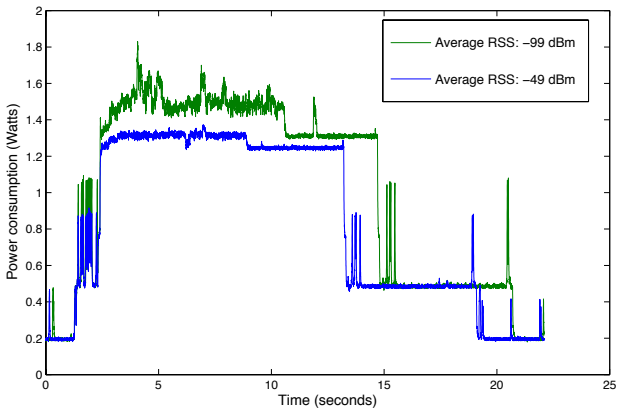

Figure 4: Power consumption at different RSS.

Our measurements show that the average transmission power during the data transmission in DCH substantially increases with a weak signal (up to 12%). The overall energy consumption is increased by 16%. The DCH_TAIL also consumes 4% more energy. The signaling traffic suffers from the same effect, increasing its power consumption up to 18%. Surprisingly, the power consumption of FACH is not increased when the RSS is weaker, which makes a call to avoid state transition to DCH instead of to FACH as much as possible. The reason behind this power increase is that to keep up with the communication the power amplifier of the terminal increases its gain to compensate for the RSS drop. Fig. 4 also shows that the download takes longer at weak RSS, which consumes more energy. The UE employs link adaptation adjusting the data rate in order to compensate for the different channel conditions. Therefore, a terminal consumes more energy under poor radio-link conditions since the transmission power is higher and the data rate is lower.

In summary, our measurements provide knowledge about the main factors that impact energy consumption of 3G in

the setting of the tests that we will perform to evaluate our algorithms in section 4. We see that there is room for an energy saving scheme based on making the UE aware of the energy consuming characteristics of the radio layer.

4. CROSS-LAYER BURST BUFFERING

Previous works in this area [3, 4, 6, 8] perform data aggregation and burst transmissions without awareness of radio layer parameters such as RLC data buffer thresholds. Obviously this is not optimal in the sense that running TailEnder or similar approaches will aggregate small data transfers causing state transitions to DCH when unnecessary.

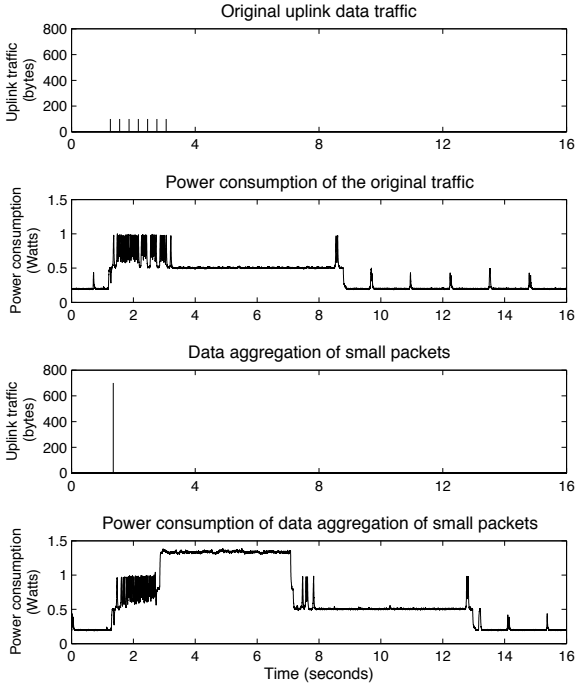

Figure 5: The effect of small packet aggregation on the energy consumption.

Fig. 5 shows an example of this event, where small data packets from a real data trace are aggregated and sent in a burst causing a state transition to DCH and more energy consumption than sending them in FACH. In the original uplink data traffic 7 packets of 100 bytes are sent every 300 milliseconds. When these packets are aggregated to send in burst they lead to 42% more energy consumption.

As we have shown in previous sections, the impact of network parameters such as tails, RLC data buffers or quality of radio links on the energy consumption of the UE is high. However, these parameters are not available for the application layers. Therefore, we have developed algorithms in order to provide estimates of the state transition timers and buffer thresholds to our scheduling algorithm. In this work we do not consider fragmentation nor the use of the radio link quality estimator like RSS to adapt our data transmissions and they are left for future works.

The structure of the section is as follows: sections 4.1 and 4.2 present the algorithms for estimation of the radio layer parameters. Our scheduling algorithm that uses these estimates is described in section 4.3.

4.1 Tail estimation algorithms

The algorithms to infer T1 and T2 (inactivity timers from Fig. 1) are based on the knowledge that each state has a qualitatively different round-trip time (RTT) for a data packet. The estimation of the inactivity timers can take place whenever a change of the network parameters is suspected. We call this a $ReconfigurationEvent$ and the discussion of the procedure to detect it is deferred to future sections. To calculate RTT, we send synthetic test packets with predetermined sizes from the UE. A server echoes the answer allowing the computation of RTT. We adopt two different approaches for estimating T1 and T2. In both estimations the UE starts in the PCH state. Before presenting the algorithm in detail (Algorithm 1), we provide an overview below.

For the DCH timer T1, we trigger a state transition to DCH by sending a large test packet followed by a sequence of small packets. At some point, the UE will switch down to FACH state due to the small packet sizes. At this time, the longer RTT will be the indicator of the state transition. Using this process we also have an indication of the RTT in the DCH and FACH states (DCH-RTT and FACH-RTT).

For the FACH timer T2, the above regime would not work. Sending small packets would keep the UE in FACH indefinitely. Therefore, we begin by a guess for lower and upper bound of T2. Note that the upper and lower bounds need not be accurate values. We can start by any guess and iteratively converge to T2 by reducing the gap between them. We start with a trial timer value between the bounds. We send a test packet that triggers FACH, wait for the trial timer and adjust the lower and upper bounds based on the UE state. If UE is in PCH, the chosen trial timer is longer than T2. If UE is in FACH, the trial timer is shorter than T2. Using binary search and a sequence of send-wait-checks we converge to an accurate value of T2. Algorithm 1 describes the two estimation procedures.

$InterPacketInterval$ and $TestPeriod$ are used for inferring the value of T1. $InterPacketInterval$ defines the time between the sending of test packets and $TestPeriod$ is the duration of estimating the value of T1. Note that this period has to be longer than the actual (unknown) T1. As described above, $LowerBound$ and $UpperBound$ are used for the binary search of the value of T2 until the gap between them is smaller than $MinGap$. Note that $UpperBound$ should be longer than the actual (unknown) T2. All values are specified in seconds.

We further describe the part of Algorithm 1 that infers T1 (lines 2-15) using an example. We define as large test packet a packet that will trigger the PCH-DCH state transition (e.g., 1200 bytes). A small test packet is the minimum size packet. Fig. 6 shows the RTT values computed by running Lines 3-12. In line 13, the $DetectRTTJump$ procedure simply identifies an RTT increase in orders of magnitude. In the example, the increase shown for packet number 9 indicates a DCH-FACH transition. Therefore, the procedure returns j equal to 9, which leads to the value of T1 to be 4.5 seconds (9 times 0.5).

Lines 16-29 infer the inactivity timer T2 by performing the aforementioned binary search. In line 17 the UE sends a small packet that triggers the transition to FACH state. The UE waits $trialTimer$ seconds and checks its state by computing the RTT. If the state is still FACH, T2 is longer than the trial time and the lower bound is updated. Other-

Algorithm 1 Inactivity timer inference

Input: InterPacketInterval, TestPeriod, LowerBound, UpperBound, MinGap
Output: T1, T2 //Inferred parameters

```
1: for each ReconfigurationEvent do
     //T1 inference. UE starts in PCH.
2:   n ← TestPeriod/InterPacketInterval
3:   for i = 1 to n do
4:     if i = 1 then
5:       send large test packet          //Force move to DCH
6:     else
7:       send small test packet
8:     end if
9:     receive echo test packet
10:    compute RTT(i)
11:    wait InterPacketInterval
12:  end for
13:  j ← DetectRTTJump(RTT)
14:  T1 ← InterPacketInterval · j
15:  FACH-RTT ← average(RTT(j + 1), RTT(n))
     //T2 inference. UE starts in PCH.
16:  while (UpperBound − LowerBound > MinGap) do
17:    send small test packet
18:    trialTimer ← (UpperBound − LowerBound)/ 2
19:    wait trialTimer
20:    send small test packet
21:    receive echo test packet
22:    compute RTT
23:    if RTT is close to FACH-RTT then
24:      LowerBound ← trialTime
25:    else
26:      UpperBound ← trialTime
27:    end if
28:  end while
29:  T2 ← trialTime
30: end for
```

wise, if the state of the UE is PCH, T2 is shorter than the trial time and the upper bound is updated. This process continues until the gap between the lower and upper bound is smaller than $MinGap$.

4.2 RLC data buffer threshold estimation

We next describe the algorithm for inferring the RLC data buffer thresholds that trigger state transitions. There are typically four thresholds, two for the PCH-DCH transition and two for FACH-DCH transition (uplink and downlink). Algorithm 2 describes our approach for inferring the PCH-DCH and FACH-DCH thresholds respectively. As in the previous algorithm, we use the RTT to infer the state of the UE by sending a small test packet and receiving the echo from the server.

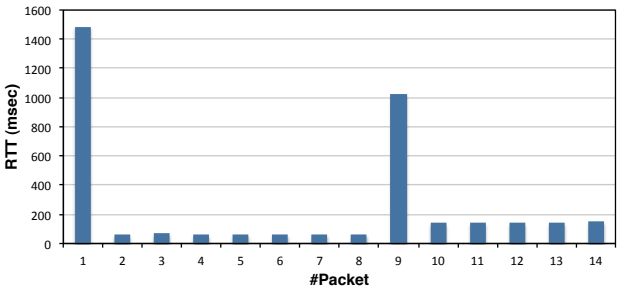

Figure 6: RTT values obtained for T1 inference. In the example InterPacketInterval is set to 0.5 seconds and TestPeriod of T1 is 7 seconds.

For inferring the PCH-DCH data buffer threshold, we use an estimate of the buffer size denoted by an upper and lower bound ($MaxBytes$ and $MinBytes$) in an analogous manner to Algorithm 1. T1 and T2, inferred by the previous algorithm, are used to define a boundary for every trial starting from PCH. This is shown in lines 1-17 of Algorithm 2.

Algorithm 2 Buffer threshold inference

Input: T1, T2, FACH-RTT
//parameters for PCH-DCH
MaxBytes, MinBytes, MinDiff,
//parameters for FACH-DCH
InterPacketInterval, LargeIncrement, SmallIncrement
Output: B_PCH-DCH, B_FACH-DCH //Buffer thresholds

```
1: for each ReconfigurationEvent do
     //PCH-DCH buffer threshold inference. UE in PCH.
2:   while (MaxBytes − MinBytes > MinDiff) do
3:     triaSize ← (MaxBytes − MinBytes)/ 2
4:     send trialSize test packet
5:     wait InterPacketInterval
6:     ComputeRTT()
7:     if RTT is close to FACH-RTT then
8:       MinBytes ← trialSize
9:       wait T2
10:    else
11:      MaxBytes ← trialSize
12:      wait T1 + T2
13:    end if
14:  end while
15:  B_PCH-DCH ← trialSize
     //FACH-DCH buffer threshold inference. UE in FACH.
16:  RTT ← FACH-RTT
17:  trialSize ← 0
18:  increment ← LargeIncrement
19:  for phase = 1 to 2 do
20:    while RTT is close to FACH-RTT do
21:      trialSize ← trialSize + increment
22:      send trialSize test packet
23:      wait InterPacketInterval
24:      ComputeRTT()
25:      wait InterPacketInterval
26:    end while
27:    increment ← SmallIncrement
28:    trialSize ← trialSize − LargeIncrement
29:    wait T1
30:  end for
31:  B_FACH-DCH ← trialSize + LargeIncrement
32: end for
```

For inferring the FACH-DCH data buffer threshold, we adopt a different approach. In this case, by triggering fewer state transitions we save energy used up for inference. This part of the algorithm works in two phases. In phase one, we approach the buffer threshold using large increments to the test packet size. In the second phase we use small increments in order to fine-tune the estimate of the threshold.

We illustrate the bottom part of Algorithm 2 that infers the FACH-DCH buffer threshold in Fig. 7. The impulses show the time and the size of the sent trial test packets (the packets to compute the RTT are omitted). The test packet size is increased in FACH by a large increment (50 bytes) until the DCH state is triggered. The 300 bytes sent in time point 5 triggers the state transition, therefore the buffer threshold is between 250 and 300 bytes. The RTT calculated with each packet is used to infer the current UE state. Then, after waiting T1 to return to FACH from DCH in time point 10, the size increment is set to $SmallIncrement$ and the algorithm starts sending 250 +

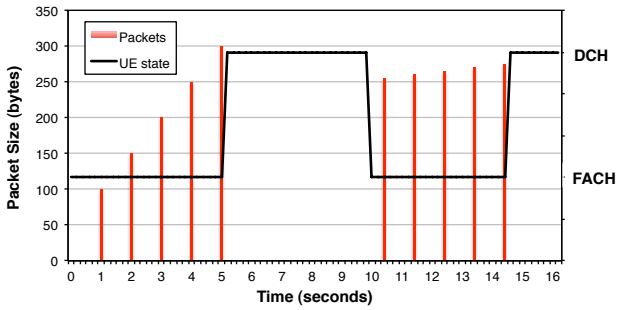

Figure 7: Example of lines 18-34 of algorithm 2 showing the packets sent, their size and the state of the UE.

increment bytes again (e.g., 260 bytes in time point 10). Note that the $InterPacketInterval$ in Algorithm 2 has to be large enough to allow the clearing of the RLC data buffer. The example uses an $InterPacketInterval$ of 1 second.

4.3 Cross-layer burst buffering algorithm

We next present our uplink scheduling algorithm that uses the parameters inferred in the previous sections. The goal of the algorithm is to minimise energy consumption by means of scheduling small data packets in FACH state and avoiding costly state transitions to DCH that lead to DCH_TAIL and FACH_TAIL energy overheads. In addition, we will stop sending packets before their deadlines while we are in the PCH state. Our algorithm is divided in two parts: a packet driven mechanism that inserts application packets in two different queues depending on their size, and a UE state based mechanism that transmits the actual packets.

Let us define the problem as follows. Let $A_h = \{a_1, ..., a_n\}$ be the set of running applications with strict time constraints (c.f. hard real-time). Let T_h be the shortest relative deadline for each packet generated by an application in the set A_h. Let $A_s = \{b_1, ..., b_n\}$ be the set of applications with soft time constraints. Let T_s be the shortest tolerable delay for scheduling transmissions for the set A_s. In practice, $T_h < T_s$. We further classify the packets of the set of applications ($A_h \cup A_s$) in two categories according to their size, L (for long) and S (for short). Let Q_S and Q_L be queues of packets queued according to their size and sorted by inserting the shortest deadline at the front of the queue. We assume that these queues are implemented using red-black trees for good performance. Algorithm 1 and 2 infer the inactivity timers T1-T2 and the RLC buffer thresholds B_PCH-DCH and B_FACH-DCH (state transition from PCH to DCH and FACH to DCH respectively). T_{clear} is the time to clear out the RLC buffers described in section 2.1 and can be measured [17].

Let us elaborate on the deadlines for the packets. T_s represents the deadline of DTA, e.g., updates from RSS, e-mail or Facebook applications. T_h is defined as a shorter deadline for other applications. It is clear that the maximum energy savings are achieved when $T_h = T_s$. Therefore, for the sake of simplicity and in order to study the maximum energy savings, we set $T_h = T_s$ for our algorithms.

Algorithm 3 describes the energy-aware burst buffering. The first part of Algorithm 3 (lines 1-7) simply queues the packets depending on their size sorted by shortest deadline. Q_S contains packets that can be sent in FACH without triggering a state transition to DCH.

The main part of Algorithm 3 (lines 8-48) transmits data

Algorithm 3 Cross-layer burst buffering

Input: B_PCH-DCH, B_FACH-DCH, T_{clear}
1: **for each** new packet p **do**
2: **if** $p.size \geq$ B_FACH-DCH **then**
3: $Q_L \leftarrow p$ sorted by shortest deadline
4: **else**
5: $Q_S \leftarrow p$ sorted by shortest deadline
6: **end if**
7: **end for**
//UE state-based packet scheduler. UE in PCH.
8: **while** $Q_L \cup Q_S \neq \emptyset$ **do**
9: **if** UE state = DCH **then**
10: transmit Q_L
11: transmit Q_S
12: **end if**
13: **if** UE state = FACH **then**
14: sum $\leftarrow 0$
15: **while** $Q_S \neq \emptyset$ **do**
16: $p \leftarrow Q_S.front$
17: sum $\leftarrow p.size + sum$
18: **if** sum $>$ B_FACH-DCH **then**
19: wait T_{clear}
20: sum $\leftarrow p.size$
21: **end if**
22: transmit p
23: **end while**
24: **end if**
25: **if** UE state = PCH **then**
26: $T_{nearest} \leftarrow$ shortest deadline $\in (Q_L \cup Q_S)$
27: wait until $T_{nearest}$
28: choose $p \in Q_L \cup Q_S$ with $T_{nearest}$
29: **if** $p \in Q_L$ **then**
30: transmit Q_L //Leads to DCH
31: **else** //Packet with shortest deadline is in Q_S
32: compute $V_L \leftarrow \sum p.size \ \forall \ p \in Q_L$
33: **if** $V_L \geq$ B_PCH-DCH **then**
34: transmit Q_L //Leads to DCH
35: **else** //Choose the channel based on Q_S
36: compute $V_S \leftarrow \sum p.size \ \forall \ p \in Q_S$
37: ch $\leftarrow$ ChannelChoice(V_S) //Function 1
38: **if** ch = DCH **then**
39: transmit Q_S //Leads to DCH
40: transmit Q_L
41: **else**
42: transmit Q_L //Leads to FACH
43: wait T_{clear}
44: **end if**
45: **end if**
46: **end if**
47: **end if**
48: **end while**

based on the UE state if there is any packet to be sent. The actions that the UE can perform are to transmit a single packet, transmit a queue in a burst or wait. The UE starts from the PCH state and for every packet sending is able to distinguish between a state transition to DCH or FACH with the knowledge of the RLC buffer thresholds and T1-T2 timers. Every time a packet is sent it updates the time of the last transmission and the current state. After the inactivity timer duration the state of the UE is changed.

When the UE is in DCH we transmit the packets stored in Q_L and Q_S. When the UE is in FACH, it transmits the packets stored in Q_S avoiding to trigger a costly state transition to DCH. This is done by sending a number of packets that sum up to less bytes than the buffer threshold. Then, the UE waits T_{clear} before the next sending of packets. This allows the UE to remain in FACH state until Q_S is emptied.

Function 1 ChannelChoice

Input: V_S //Volume of Q_S in bytes
Output: channel
  P_1, P_2 //Power consumption of DCH and FACH
  T1, T_{clear}, B_FACH-DCH
  if $V_S < P_1/P_2 \cdot$B_FACH-DCH$\cdot$T1$/T_{clear}$ **then**
    channel $\leftarrow$ FACH
  else
    channel $\leftarrow$ DCH
  end if

Finally, when the UE is in PCH, the algorithm compares the deadlines of packets in the front of Q_L and Q_S and waits for the nearest deadline $T_{nearest}$. Next, the UE decides if it will send a packet or burst that triggers a state transition to DCH or FACH. There are three cases.

First, in case the packet with the shortest deadline is in Q_L (line 29), it needs to be sent. Q_L will be transmitted triggering a transition to DCH. Second, if the packet is in Q_S, the UE will transmit Q_L if the volume in bytes of Q_L is larger than the PCH-DCH buffer threshold (line 31-34). This will lead to a trigger of a transition to DCH.

In the third case, the transmission in FACH is feasible with the current volume of the queues. However, a large volume of Q_S can lead the UE to stay in FACH for a long period, which would consume more energy than sending the queues in burst in DCH. The choice of the less consuming option, i.e., sending on DCH or FACH, is performed by Function 1 ($ChannelChoice$) in line 37 given the volume of Q_S. If sending in FACH is less consuming, the UE will transmit Q_L which triggers a FACH transition and wait T_{clear}. Q_S will be emptied in the FACH state. If sending in DCH is less consuming, the UE transmits both queues in burst triggering a DCH transition.

Function 1 is an optimisation to avoid the case when a large volume of data in Q_S (denoted by V_S) leads the UE to stay in FACH for a long period. Given the power consumption of DCH and FACH (P_1 and P_2) and the transmission time (ϵ) of Q_S in DCH, the energy cost of transmitting in DCH is estimated by $E_1 = P_1(\text{T1}+\epsilon)+P_2\text{T2}$. Given the high data rate of DCH, we consider that ϵ is negligible for the estimation. The energy cost in FACH is estimated by $E_2 = P_2(\text{T2}+T_{clear} \cdot V_S/\text{B\_FACH-DCH})$. Therefore, when the inequality $E_1 > E_2$ holds, $ChannelChoice$ will return FACH, otherwise DCH.

5. EVALUATION

We begin this section by presenting the evaluation of the inference algorithms followed by the evaluation environment for the scheduling algorithm and discussing the results.

5.1 Evaluation of inference algorithms

We implemented Algorithms 1 and 2 in Java using UDP packets to calculate the RTT. A simple UDP server was developed to echo UDP packets. The algorithm to infer the buffer thresholds (Algorithm 2) was also implemented in the UDP server to infer the downlink thresholds.

Algorithm 2 was evaluated using our measurement setup described in section 3.1. We first ran the algorithm and inferred the thresholds. In order to verify them, we sent packets with the triggering sizes and observed the state transitions on the live power consumption trace generated by our measurement setup. The settings for the test were 1100 bytes for $MaxSize$, 50 bytes for $MinSize$ and 10 bytes for $MinDiff$. $LargeIncrement$ and $SmallIncrement$ were set to 50 and 2 bytes respectively. T1 and T2 were set to 5. $InterPacketInterval$ of Algorithm 2 was set to 1.5 seconds. Averaging 3 rounds of the running of the algorithm itself consumed 78 Joules and took 97 seconds.

Algorithm 1 was evaluated in a similar manner. We compared the values inferred by the algorithm to the time that the UE stays in DCH and FACH (observed in the recorded power trace) when triggering a PCH-DCH transition. The settings used in the implementation of Algorithm 1 were the following: small and large tests packets were set to 43 and 1200 bytes respectively. $InterPacketInterval$ was 0.5 seconds. $InitialGuess$, $LowerBound$, $UpperBound$ and $MinGap$ were set to 7, 0, 16 and 0.5 seconds for the tests respectively.

We also compared our algorithm to the one presented by Qian et al. [18] in terms of energy consumption and elapsed time. In short, the idea behind their state demotion inference algorithm is to use the RTT difference between states to infer the state transitions and therefore inactivity timers. Their algorithm sends a packet that triggers PCH-DCH state transition and calculates the RTT by sending another packet after a number of seconds. They increase the number of seconds starting from 0 to a maximum and infer the inactivity timers from the measured RTT values. The elapsed time and the energy consumption of the algorithm is very dependent on the first guess of the maximum. It is clear that the maximum needs to be longer than the sum of T1 and T2. We implemented their algorithm in Java using UDP packets. For comparing with our algorithm we used the same value for $TestPeriod$ as their maximum, and started our $UpperBound$ with $TestPeriod$. The average of 3 different rounds is calculated for each of the points in Fig.8.

Fig. 8 shows that their algorithm results in larger elapsed time and more energy consumption. It can be seen that whereas the increase of consumed energy and elapsed time is somehow linear in their case, in our case it varies depending on the performance of the binary search (changing the maximum, we change the upper bound of the search). However, in the least favourable case, our algorithm consumes 65% less energy and takes 35 seconds less to converge on the estimated value for the timers.

5.2 Evaluation environment for scheduler

The evaluation environment consists of the physical measurement setup and a UDP server on the Internet. The UDP server was used to keep track and make sure that all the intended packets were indeed sent from our physical measurement setup. We developed a C++ TestFramework which runs in the test computer of our physical setup in order to run any packet-driven algorithm. The application allows us to replay any previously captured data traffic trace and schedule the packet sendings to our 3G modem.

The traces are pcap (packet capture) files that consist of data traffic packets. An input parser uses *TShark*[2] commands to convert the gathered pcap traces into sequence of packet sendings with the following format: {*timestamp*, *packetsize*, *packetnumber*, *source*, *destination*, *protocol*}.

An user thread replays the original trace providing the input packets to the interchangeable algorithm in turn. We

[2]Terminal based Wireshark.
http://www.wireshark.org/docs/man-pages/tshark.html

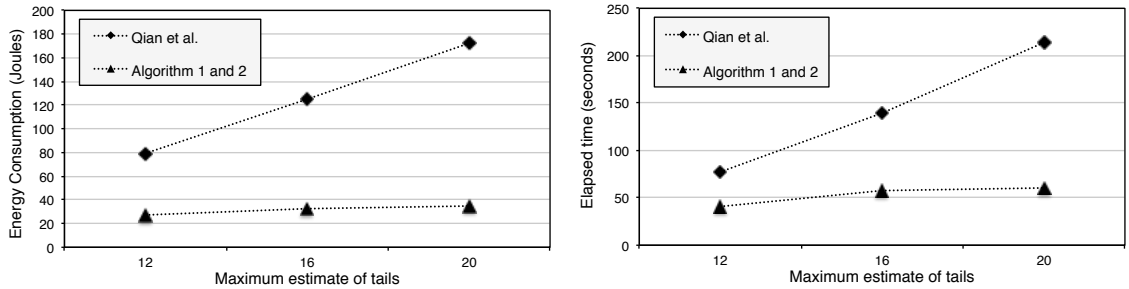

Figure 8: Comparison of algorithms in terms of energy consumption and elapsed time.

implemented a simple packet forwarder (simulating FIFO), our cross-layer burst buffering algorithm and the scheduling algorithm of TailEnder [3] for DTA. Our algorithm is implemented in two different threads (producer-consumer). The queues are implemented using C++ STL multimaps.

Our TailEnder implementation is divided in two threads as well. The packet driven thread sends or queues a packet based on the following: if the UE is still within a time fraction of the tail time from the last sending it directly sends the new packet, otherwise the packet is queued. The other thread checks the shortest deadline of the queue and schedules transmissions when it is about to expire. Our implementation keeps track of the fraction of tail time by calculating the relative elapsed time from the last transmission.

Finally, we use a network module which provides the algorithms the functionalities to send and receive packets to the server using simple functions based on UDP sockets.

5.3 Data generation

In the data gathering phase, we used *tcpdump* in a Samsung Galaxy SII running Android 2.3 connected to the same operator in Sweden using the SIM card described in Section 3. In order to only allow the traffic from the desired applications we used a *iptables* based firewall named DroidWall[3]. We captured all the 3G communication for 40 minutes.

The applications running in the smartphone were Skype version 2.5.0.108 and Facebook 1.7.2. All the traces were gathered with the screen switched off and the application running in background. No user interaction was performed. The gathered trace has many small packets.

5.4 Energy saving results

In order to calculate the energy savings, we transmitted the original trace gathered from the Android smartphone using the TestFramework and measuring the energy consumption by means of our physical setup. The energy consumption value obtained from the trace is used as base line for both algorithms. For each of the points in Fig. 9 we ran our TestFramework with the original trace as input and measured the energy consumption using TailEnder and our cross-layer burst buffering algorithm separately. Then we computed the energy savings with respect to the base line (i.e., no burst buffering) varying the deadline for the packets. We set T_{clear} to be a constant with value 300 milliseconds. The energy consumption base line was 777 Joules for the original trace.

Fig. 9 shows that the energy savings of the cross-layer scheduler are higher than TailEnder. The minimum energy savings in our measurements are 27%.

[3]http://code.google.com/p/droidwall/

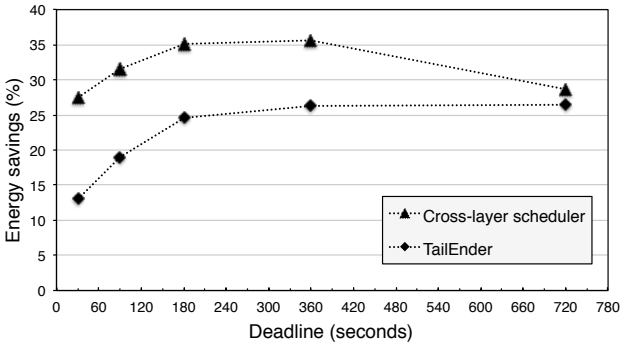

Figure 9: Energy savings of cross-layer scheduling and TailEnder.

It is worth mentioning that for the shortest deadline our method achieves higher energy savings compared to TailEnder. This is due to the fact that transmitting small packets in FACH is more energy efficient than triggering DCH state transitions. Our method takes advantage of the small packets avoiding the unnecessary state transitions to DCH.

For TailEnder, the larger the deadline the more energy savings can be achieved. This is due to the fact that the traffic aggregation is performed with more data packets and then triggering a DCH state transition is needed. The same issue arises for the cross-layer burst buffering when increasing the deadline. However, the energy savings of our method decreases for applications with longer deadlines. This happens since the probability of having bigger packets in the queues increases when increasing the deadline and therefore the possibilities of sendings in FACH are fewer.

The gap in Fig. 9 can further increase by network settings that have higher B_FACH-DCH or shorter T_{clear}. Further, data sets with smaller packets would be more beneficial to our system; as it would be the case with a longer T1. However, our experiments confirm that the gap will not be reversed even with parameter changes in the network settings.

6. CONCLUSION

In the search for reducing the energy consumption in wireless networks awareness of the radio communication layer plays an important role. In our work, we have quantified the energy footprint characteristics of the 3G radio communication layer at the user end by performing actual physical measurements on a modern broadband module, isolating the energy consumption of data transfers from other energy consuming factors (e.g., CPU usage or screen rendering). In particular, we show that the inactivity timers of the RRC state machine, RLC data buffer thresholds and radio coverage of the network drastically influences the energy consumption of the 3G data transfers.

The knowledge of the hidden energy footprint characteristics of the radio communication layer becomes valuable when developing energy efficient solutions to optimise the usage of the battery resource. We provide the upper layers with the needed information about the parameters that influence the 3G energy consumption. To support this a set of algorithms that are able to infer the parameters and a novel uplink scheduling algorithm are proposed. Our scheduler uses a more detailed radio communication layer knowledge to improve the energy savings compared to state-of-the-art. The results of our work demonstrate that the radio communication energy footprint awareness is fundamental for achieving higher energy savings.

Our current work includes implementing a middleware based on our scheduling algorithm as a kernel module for Android. Extensions of the work include explorations with other types of applications and the inclusion of radio link quality awareness. Furthermore, based on the results in this paper, we believe that there is room for an energy saving scheme based on fragmentation in order to take advantage of FACH data transfers.

Acknowledgements

This work was supported by the Swedish national graduate school in computer science (CUGS). The authors wish to thank the support of Ericsson AB, and in particular B-O Hertz, Pär Emanuelsson and Claes Alströmer for providing the measurement kit and facilitating the measurement gathering phase.

7. REFERENCES

[1] T. Armstrong, O. Trescases, C. Amza, and E. de Lara. Efficient and transparent dynamic content updates for mobile clients. In *Proc. of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, 2006.

[2] M. Asplund, A. Thomasson, E. J. Vergara, and S. Nadjm-Tehrani. Software-related energy footprint of a wireless broadband module. In *Proc. of the 9th ACM international symposium on Mobility management and wireless access*, MobiWac '11, 2011.

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

[4] M. Calder and M. Marina. Batch scheduling of recurrent applications for energy savings on mobile phones. In *7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks,* SECON 2010.

[5] I. Humar, X. Ge, L. Xiang, M. Jo, M. Chen, and J. Zhang. Rethinking energy efficiency models of cellular networks with embodied energy. *IEEE Network,*, 25(2):40-49, 2011.

[6] V. Könönen and P. Paakkonen. Optimizing power consumption of always-on applications based on timer alignment. In *3th International Conference on Communication Systems and Networks,* COMSNETS 2011.

[7] H. Lagar-cavilla, K. Joshi, A. Varshavsky, J. Bickford, and D. Parra. Traffic Backfilling: Subsidizing Lunch for Delay-Tolerant Applications in UMTS Networks. In ACM MobiHeld, 2011.

[8] H. Liu, Y. Zhang, and Y. Zhou. Tailtheft: leveraging the wasted time for saving energy in cellular communications. In *Proc. of the sixth international workshop on MobiArch*, MobiArch 2011.

[9] L. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. Annual IEEE International Conference on Computer Communications*, INFOCOM 2001.

[10] M. Marsan, L. Chiaraviglio, D. Ciullo, and M. Meo. Optimal energy savings in cellular access networks. In *IEEE International Conference on Communications*, (ICC Workshops) 2009.

[11] Z. Niu, Y. Wu, J. Gong, and Z. Yang. Cell zooming for cost-efficient green cellular networks. *IEEE Communications Magazine*, 48(11):74-79, 2010.

[12] G. Perrucci, F. Fitzek, G. Sasso, and M. Katz. Energy Saving Strategies for Mobile Devices using Wake-up Signals. In *4th International Mobile Multimedia Communications Conference* , MobiMedia 2008.

[13] G. P. Perrucci, F. Fitzek, G. Sasso, W. Kellerer, and J. Widmer. On the impact of 2G and 3G network usage for mobile phones' battery life. In *European Wireless*, 2009.

[14] RLC protocol. 3gpp specification 25.322. http://www.3gpp.org/ftp/Specs/html-info/25322.htm

[15] I. Puustinen and J. Nurminen. The effect of unwanted internet traffic on cellular phone energy consumption. In *International Conference on New Technologies, Mobility and Security*, NTMS 2011.

[16] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *18th IEEE International Conference on Network Protocols*, ICNP 2010.

[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 *Proc. of the 9th international conference on Mobile systems, applications, and services*, 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 *Proc. of the 10th annual conference on Internet measurement*, IMC 2010.

[19] A. Rice and S. Hay. Measuring mobile phone energy consumption for 802.11 wireless networking. *Pervasive Mob. Comput.*, 6:593–606, 2010.

[20] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proc. of the sixteenth annual international conference on Mobile computing and networking,* MobiCom 2010.

[21] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *Proc. of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT 2009.

[22] E. J. Vergara, S. Nadjm-Tehrani, M. Asplund, and U. Zurutuza. Resource footprint of a manycast protocol implementation on multiple mobile platforms. In *Proc. of 5th International Conference on Next Generation Mobile Applications, Services and Technologies,* NGMAST 2011.

[23] L. Wang and J. Manner. Evaluation of data compression for energy-aware communication in mobile networks. In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, CyberC 2009.

[24] L. Wang and J. Manner. Energy consumption analysis of wlan, 2g and 3g interfaces. In *Proc. of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM 2010.

[25] Y. Xiao, R. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *Proc. of 2nd International Conference on Next Generation Mobile Applications, Services and Technologies,* NGMAST 2008.