Performance-aware Energy Optimization on Mobile Devices in Cellular Network

Yong Cui, *Member, IEEE,* Shihan Xiao, Xin Wang, *Member, IEEE,* Zeqi Lai, Zhenjie Yang, Minming Li, *Senior Member, IEEE,* and Hongyi Wang

Abstract—In cellular networks, it is important to conserve energy while at the same time satisfying different user performance requirements. In this paper, we first propose a comprehensive metric to capture the user performance cost due to task delay, deadline violation, different application profiles and user preferences. We prove that finding the energy-optimal scheduling solution while meeting the requirements on the performance cost is NP-hard. Then we design an adaptive online scheduling algorithm PerES to minimize the total energy cost on data transmissions subject to user performance constraints. We prove that PerES can make the energy consumption arbitrarily close to that of the optimal scheduling solution. Further, we develop offline algorithms to serve as the evaluation benchmark for PerES. The evaluation results demonstrate that PerES achieves average 2.5 times faster convergence speed compared to state-of-art static methods, and also higher performance than peers under various test conditions. Using 821 million traffic flows collected from a commercial cellular carrier, we verify our scheme could achieve on average 32%-56% energy savings over the total transmission energy with different levels of user experience.

Index Terms—Mobile cellular network, Energy-efficiency, Performance aware

1 Introduction

There is a quick growth of cellular applications in recent years thanks to the constant increase of the processor power of mobile devices and the transmission bandwidth of cellular networks. The capacity of batteries, however, grows at a much slower speed and the limited battery life has become the bottleneck that prevents the support of advanced mobile applications.

Energy conservation is often supported by existing wireless MAC protocols. Specifically, with radio resource control (RRC) in UMTS (Universal Mobile Telecommunications System) network, a radio does not turn to a low power state immediately after data transmissions, but instead stays at the high power state and waits for the expiration of an inactivity timer. If no transmission occurs during that period, it will switch to the low power state. This period is defined as *tail time* and the corresponding energy consumption is called *tail energy*. The tail time is designed in 3G radio access network to avoid the high signaling overhead [1], [2], and also introduced in 4G LTE networks [3] recently. A long duration of tail time, however, will compromise the energy-efficiency.

Some recent efforts have been made to reduce the tail time. The under-layer methods utilize the fast dormancy [4], [5], [6], [7] option proposed in 3GPP specifications to optimize RRC configurations [8], [9]. Application-layer solutions [10], [11], [12], [13] attempt to better schedule data transmission to reduce the tail energy. The work in [14], [15], [16], [17] investigate the varying signal impact on the *data transmission energy* in the mobile environment through both experiments and strategy design. Authors in [18], [19], [20] exploit the heterogeneity of WiFi and 3G network access to optimize the overall transmission energy. Based on existing solutions, we identify three major challenges in optimizing the energy in cellular networks.

First, there exist scheduling conflicts between the tail energy reduction and data transmission energy optimization. Tail energy can be reduced by queuing tasks and transmitting data in batch, while data transmission energy may be reduced if data are transmitted upon good channel condition. It may not be easy to find the optimal waiting time to meet both requirements and minimize their total energy. The literature work often focuses on reducing one type of energy only, while it is important to reduce both types of energy for an overall lower system energy consumption. We call the two types of energy together as *hybrid energy*.

Second, it is difficult to satisfy specific user transmission requirements while at the same time minimizing the energy consumption. The attempts to reduce the hybrid energy often introduce delay in packet transmissions. User expectation on performance depends on many factors, such as the task types, application profiles, and user preferences. Assuming that the delay has exactly the same impact on the user transmission experience regardless of the application or time, existing efforts on energy conservation fail to address specific or dynamic user application requirements. For example, a businessman can not tolerate a big Email delay, while delay is not a big concern for a student. Further, for a given user, the requirement varies depending on the application or time. It remains a challenge to characterize the flexible user requirements and fully explore the differences

- *Y. Cui, S. Xiao, Z. Lai, Z. Yang and H. Wang are with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, P.R.China.*
  E-mail: cuiyong@tsinghua.edu.cn, xiaosh12@mails.tsinghua.edu.cn
- *X. Wang is with Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, New York, USA. Email: x.wang@stonybrook.edu.*
- *M. Li is with Department of Computer Science, City University of Hong Kong, Hong Kong, China. Email: minming.li@cityu.edu.hk.*

in user expectations on applications to minimize the hybrid energy consumption. Moreover, it is critical for a scheduling scheme to quickly adapt to changing user requirements to better save energy.

The third issue is the feasibility in real systems. Schemes that need lower-layer system support such as optimizing the timer of RRC often require modifying configurations controlled by cellular network carriers, which makes practical implementation difficult. Also, a transmission scheduling mechanism often needs to deal with transport layer (e.g., the congestion control and flow control of TCP) and application layer (e.g., interfaces exposed to applications) protocols and configurations.

To the best of our knowledge, this is the first work that shows how to optimize the hybrid energy consumption while meeting user transmission requirements on different applications. Our main contributions are summarized as follows:

- We model the *performance-aware hybrid energy optimization problem* in mobile cellular networks and prove its NP-Hardness, and design offline solutions to serve as the performance benchmarks.
- Based on a novel queueing method, we design an online scheduler to meet varying user requirements on application performance while ensuring the energy conservation.
- We propose an adaptive method for the online scheduler to converge more quickly to the optimal energy bound compared to state-of-art static strategies, and our performance studies demonstrate that a larger amount of energy can be saved when the user requirements change over time.
- Our scheme can be implemented as an application for traffic management on mobile devices, thus all the benefits can be achieved by solely upgrading the software.

The remainder of this paper is organized as follows. We introduce the related work in Section 2. Section 3 illustrates our basic problem formulation. In Section 4, we first present our scheduling analysis and solution design in the online setting, and then provide offline solutions as the evaluation benchmarks. We evaluate the performance of our scheduling algorithms in Section 5, and conclude our work in Section 6.

2 Related Work

In cellular networks, the tail energy is identified as one of the key energy problems in data transmission [1], [10], [13], [21] with three major types of solutions: 1) the timer optimization which adapts the RRC configurations to dynamic and complex traffic patterns [8], [9], but it requires modifying configurations controlled by cellular network carriers; 2) the fast dormancy strategy for mobile devices to proactively demote the transmission to a low power state by optimizing the inactivity timer based on traffic predication [4], [5], [7]; and 3) the delayed transmission, where mobile devices queue tasks and schedule transmission in batch to reduce the total tail energy [10], [11], [12], [13], [16], [17], [19], [20].

Conventional energy conservation schemes are mostly based on traffic and signal predication. The high dynamics in user traffic and wireless link quality as well as the user mobility [16], [17], however, make these schemes difficult to apply in a practical wireless network. To overcome the limitation, the *Lyapunov control theory* has been introduced in recent work for non-predication-based online scheduling [19], [20], [22], [23]. *SALSA* [19] proposes a general multi-interface online scheduling algorithm based on Lyapunov optimization, taking into account both the delay impact and wireless link quality. However, the tail energy in cellular networks is not modeled. In *eTime* [20] and *eTrain* [24], a similar Lyapunov-based scheduling method is proposed for 3G and WiFi networks with the tail energy embedded in the 3G power model. Liu et.al. also leverages the cloud computing in both theory and experiments to manage data transmissions for mobile applications [25], [26].

Recent Lyapunov-based solutions mainly face two challenges: 1) A single FIFO (First-In-First-Out) queue is applied to handle all transmission tasks without considering different task deadlines and application profiles; 2) There is no consideration of different user requirements on application delay and the potential change of requirements over time. To achieve the balance between energy and delay, cellular traffic can also be offloaded to WiFi network or neighbouring nodes through peer-to-peer interfaces (e.g., the WiFi direct). Hu et al. [27] first propose to explore the difference in data throughput between nearby nodes to offload the traffic. Rebecchi et al. give a comprehensive survey in [28] on equipping today's cellular networks with offloading capabilities.

To address above issues, our work aims to optimize the hybrid energy while satisfying dynamic user performance requirements. We propose a novel method to manage multiple delay-level queues in the Lyapunov optimization framework. Generally, a user up-link transmission request is the key trigger for the subsquent down-link data transmission [16], which allows the mobile device to control the data transmissions. Thus, our work focuses on the scheduling of up-link traffic, as done in previous work [4], [5], [7], [10], [11], [12], [13], [16], [17], [19], [20], [21].

3 Problem Formulation

The principle of our design is to aggregate and schedule the traffic generated by different applications. By postponing and batching transmissions judiciously, we can achieve the performance-aware energy optimization. In this section, we will elaborate on how we handle the performance and energy issues and formulate the transmission scheduling problem.

3.1 Motivating Example

We show an example to illustrate the trade-off between energy saving and user performance requirement in Fig. 1. The data rate over 3G network varies over time (every tick on the x-axis marks a 10 second interval). Three application tasks (APP1, APP2 and APP3) arrive at ticks 0, 6 and 10 with the data sizes 1.25MB, 0.25MB and 4.25MB respectively. The power consumption of the 3G interface on the mobile

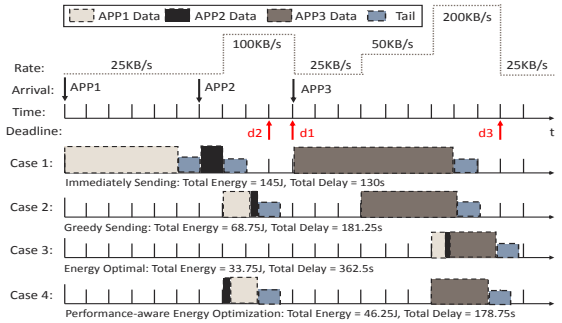


Fig. 1: An example of the performance-aware hybrid energy optimization

phone is set to 1W, and the tail time is 10 seconds with the tail power being 0.5W. Since a user may have different expectations on application performance, we call the user performance requirement as a tolerable delay whose value is set to be a fraction of the corresponding task deadline. In this example, we assume the user have the highest expectation on APP2 and the lowest interest in APP1, i.e., the fractions for APP1, APP2 and APP3 are 100%, 50% and 80% respectively.

In Fig. 1, we show the energy and delay performance of four transmission scheduling strategies. *Immediately-Sending* transmits the tasks immediately after they arrive, and thus achieves the minimum delay. Taking APP1 as an example, the first 5 ticks are used for data transmission and the tail energy is consumed in the next 1 tick, so the total energy cost of APP1 is $0.5W \times 1tick \times 10s/tick + 1W \times 5tick \times 10s/tick =$ 55J. The energy consumption of other applications are calculated in the same way. The delay for each application is defined as the duration between the time of the application arrival and the time of the transmission completion. *Greedy-Sending* maintains a FIFO queue to buffer the arriving tasks, and transmits data only when the rate is larger than 25KB/s to take advantage of the physical channel conditions to conserve energy. However, the waiting delay makes APP2 violate its user delay requirement. *Energy-Optimal* transmits all the tasks during the highest data rate, and thus achieves the minimum energy consumption. However, it violates the task deadlines of both the APP1 and APP2 and also the user delay requirement on APP3.

The fourth strategy first transmits packets from APP2 ahead of APP1 to satisfy its user delay requirement. When APP3 arrives, it tries to wait for a better link quality within the time duration allowed by the user requirement. It meets the deadline demands of all the tasks and also satisfies the user performance requirement. Further, its total delay is half that of *Energy-Optimal* while obtaining energy saving comparable to *Energy-Optimal*, and its energy consumption is only 32% that of *Immediately-Sending*.

This example indicates that conventional FIFO-queue-based scheduling schemes adopted in existing work [19], [20] can not well address issues associated with different task delays and deadlines of multiple applications. Moreover, there is a need for the scheduler to effectively react to the given user performance constraint while conserving the energy.

3.2 Performance Impact

Motivated by the above example, we first characterize the performance impact due to the task deadlines, application profiles and user preferences. Our transmission framework runs as a daemon to collect the traffic generated by different applications, and the traffic is scheduled to transmit in the unit of time slot. The traffic from an application can be divided into multiple transmission tasks, where typically a task corresponds to a packet. For a task u, the time slot it arrives is $t_a(u)$, and the slot scheduled for it to send to the corresponding socket is $t_s(u)$. The *buffer delay* due to the scheduling is denoted as $\mathcal{D}_b(u) = t_s(u) - t_a(u)$, and the *transmission delay* cost to transmit a task u is $\mathcal{D}_t(u)$. The ending slot that u is transferred completely is denoted by $t_e(u) = t_s(u) + \mathcal{D}_t(u)$, and the deadline of u is $t_d(u)$.

We embody the bandwidth of the cellular network with the capacity of a time slot, i.e., the maximum amount of data in bytes that can be transmitted between mobile devices and the base station in one slot. Let $c(t)$ denote the capacity of a slot t and $v_u(t)$ denote the data transfer rate of a transmission task u in the slot t. The condition

$$\sum_{u \in \{u | t_s(u) = t\}} v_u(t) \leq c(t) \tag{1}$$

should be met for any slot. For clarity, we list all the related notations in Table 1.

To capture the performance impact, we introduce a performance cost metric $\phi_u(\cdot)$ by taking into account the views from three parties.

- **Task View** Different tasks generally have different delay tolerance. The task performance may degrade significantly if its delay expectation is violated. For example, the deadline of downloading video frames can be found in existing online video applications to ensure user watching experience [16]. We take the term *deadline* as the tolerable bound of the sum of buffer delay and transmission delay.
- **Application View** Different application types have different sensitivity to the delay, and the performance change with the delay can be captured by a *profile*. For example, the performance of an application may reduce linearly as the delay increases.
- **User View** Mobile users may have different performance expectation on different types of applications,

TABLE 1: Basic Notations and Definitions

Notations	Definitions
$t_a(u)$	Arrival time of task u
$t_s(u)$	Scheduled time of task u
$t_d(u)$	Deadline of task u
$t_{dd}(u)$	Maximum tolerable delay of task u
$t_e(u)$	Ending transmission time of task u
$\mathcal{D}_b(u)$	Buffer delay of task u
$\mathcal{D}_t(u)$	Data transmission delay of task u
$\mathcal{D}(u)$	Total delay of task u, i.e., $\mathcal{D}_b(u) + \mathcal{D}_t(u)$
$\mathcal{S}(u)$	Data size of task u
w_u	User preference weight of task u
$f_p(\cdot)$	Application profile function
$\phi_u(\cdot)$	Performance degradation function of task u
$P_{sig}(\cdot)$	Average power to generate a given signal
$R_{sig}(\cdot)$	Rate function that maps a received signal to the average data transmission rate
$t_s(U)$	Scheduled slot set of tasks in U, i.e., $\{t_s(u) \mid u \in U\}$

and a *weight* factor can represent a user's preference level.

We can see the delay will impact different parties, and the design should reduce the delay impact for all. If considering tasks from only one application, the data size should be taken into account because delay will increase with the traffic size. For different types of applications such as VoIP or photo-uploading, their profiles should capture their different sensitivity to the task delay. Finally, the user preference is accounted with an input weight. Hence the *performance degradation function* ϕ_u can be computed as:

$$\phi_u(\mathcal{D}(u)) = w_u \cdot f_p(\mathcal{D}(u)) \cdot \mathcal{S}(u) \tag{2}$$

where $\mathcal{D}(u) = t_e(u) - t_a(u)$ is the task delay, and $\mathcal{S}(u)$ denotes the data size in bytes of the task u. The *profile function* f_p represents the sensitivity of an application to the delay, and the *weight* w_u represents the user's preference on the application that generates u.

We can easily get the following property:

Property 1. Any $\phi_u(\cdot)$ should satisfy the following properties:

- $\phi_u(0) = 0$
- If $d_1 < d_2$, then $\phi_u(d_1) \leq \phi_u(d_2)$
- If $d_1 \leq t_d(u) - t_a(u) < d_2$, then $\phi_u(d_1) < \phi_u(d_2)$

The first two conditions ensure that the function ϕ_u captures the non-decreasing feature between the performance cost and task delay. With the performance cost being the *weighted* product of delay and data size, the third condition reflects the cost associated with the deadline violation, i.e., the user has significantly worse experience thus higher performance cost.

Let U denote the pending transmission task set, and $t_s(U)$ denote the set of scheduled slots of all the tasks in U, i.e., $t_s(U) = \{t_s(u) | u \in U\}$. Given $\phi_u(\cdot)$ for all the tasks in U, we can evaluate the total performance cost by the schedule $t_s(U)$ as $\sum_{u \in U} \phi_u(\mathcal{D}(u))$.

3.3 Energy Consumption for Data Transmission

For any given data size, the *data transmission energy* depends on the product of two factors: the transmission power and the time taken to transmit one bit of data. Previous works [14], [16], [29] have already illustrated how these factors vary with the signal strength. We take the RSSI (Received Signal Strength Indicator) value, i.e., the signal strength to evaluate the wireless link quality as it can be easily acquired on modern mobile devices without additional cost. We define the signal strength as a time-varying function $signal(t)$. The function $R_{sig}(signal)$ maps a received signal strength to the average data rate value, and the power to generate the signal is denoted by $P_{sig}(signal)$.

Suppose two data units u_1 and u_2 that already arrived in buffer are scheduled to transfer one after the other, the total data transmission time consumed is $\mathcal{D}_t(u_1) + \mathcal{D}_t(u_2)$ where each data unit takes the complete bandwidth resource for its transmission. Alternatively, the two can be transmitted concurrently with each using a fraction of the bandwidth. In this way, the total data transmission time will remain the same. Since the wireless bandwidth is fully applied to transmit the same amount of bytes in both cases, the total energy consumption also remains the same. However, the completion time of the first task gets extended while that of the second task remains the same. Therefore, with the same amount of energy consumed, in order to reduce the task completion time, it is more efficient to transmit all the data units sequentially. That is, for any feasible schedule $t_s(U)$, when we sort u in U in the ascending order by $t_s(u)$, the current task u_i should be scheduled to transmit only after the previous task u_{i-1} completes its data transmission, i.e., $t_s(u_i) > t_e(u_{i-1}),\ 2 \leq i \leq n$.

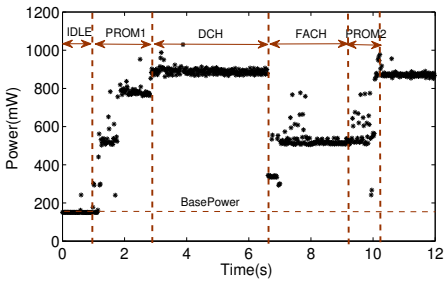


Fig. 2: Device Power at Different RRC States

Denote $\tau \in N^+$ as a discrete time slot and Δt_0 as the time length of one slot. Since the data of u will be transferred at rate $c(\tau)$ in each slot τ from $t_s(u)$ to $t_e(u)$, we have the data size of the task u as

$$\mathcal{S}(u) = \sum_{\tau=t_s(u)}^{t_e(u)} c(\tau)\Delta t_0 = \sum_{\tau=t_s(u)}^{t_e(u)} R_{sig}(signal(\tau))\Delta t_0 \tag{3}$$

In case transmission cannot be carried in some time slots, for example due to the dramatic drop of signal strength upon user movement, some slots in the above equation may have zero-rate transmissions, which naturally increases the transmission delay.

Let $E_{trans}(t_1, t_2)$ denote the *data transmission energy* consumed from the slot t_1 to t_2. If the data is transferred by consuming the transmission power $P_{sig}(signal(\tau))$ of the wireless interface in each slot τ from t_1 to t_2, the data transmission energy can be computed as

$$E_{trans}(t_1, t_2) = \sum_{t_1 \leq \tau \leq t2} P_{sig}(signal(\tau))\Delta t_0 \tag{4}$$

During time period Γ, the total data transmission energy to transmit U by a schedule $t_s(U)$ can be estimated as

$$\widetilde{E}_d(U, t_s(U), \Gamma) = \sum_{u \in U} E_{trans}(t_s(u), t_e(u)) \tag{5}$$

3.4 Tail Energy Consumption Estimation

In a UMTS network, radio resources are managed through RRC and a state machine is maintained for the radio. The state machine has three basic states: IDLE, DCH and FACH.

TABLE 2: Parameters of Different RRC States

	DCH	FACH	PROM1	PROM2
p(mW)	732.826	388.880	557.708	423.625
δ(s)	3.287	4.024	2.114	1.039

Their radio power is denoted as p_I, p_D and p_F respectively. The transition among different states is mainly determined by the data traffic conditions [8]. If the radio is at IDLE or FACH, the arrival of a data transmission unit will trigger it to promote to a higher power state DCH (the transitions IDLE→DCH and FACH→DCH are called PROM1 and PROM2 respectively, see Fig. 2). If there are no transmission tasks arriving and the radio remains inactive for a time duration, it will lead to an instant state demotion, either DCH → FACH or FACH → IDLE. The *tail energy* wasted during both the time durations (denoted by δ_D and δ_F respectively) can significantly impact the total energy consumption.

Following the measurement schemes proposed in [8], we estimate these parameters through experiments in a UMTS network in China (CHN-CUGSM) using a Google Nexus S smartphone. Table 2 lists the detailed measurement results. We take power at the IDLE state as the base in Fig. 2 and obtain the radio power in Table 2 by excluding the base power.

Let T denote one complete tail time, i.e, the sum of δ_D and δ_F. Let the set of transmission tasks $U = \{u_i | 1 \leq i \leq n\}$ be sorted in the ascending order by $t_s(u)$ of each transmission unit u. The total tail energy consumption during the transmission of U according to a schedule $t_s(U)$ within a given period Γ can be estimated in a fine-grained way as

$$\widetilde{E}_t(U, t_s(U), \Gamma) = \sum_{2 \leq i \leq n} E_{tail}(\Delta t_i) + E_{tail}(T) \quad (6)$$

where $\Delta t_i = t_s(u_i) - t_e(u_{i-1})$ and

$$E_{tail}(\Delta t_i) = \begin{cases} p_D \Delta t_i & \text{if } 0 \leq \Delta t_i \leq \delta_D \\ p_D \delta_D + p_F(\Delta t_i - \delta_D) & \text{if } \delta_D < \Delta t_i \leq \delta_D + \delta_F \\ p_D \delta_D + p_F \delta_F & \text{otherwise.} \end{cases} \quad (7)$$

3.5 Optimization Problem

We convert the transmission scheduling to an optimization problem. Given a set of pending transmission tasks $U = \{u_1, u_2, \cdots, u_n\}$, the purpose of scheduling is to determine the time slot assignment $t_s(U) = \{t_s(u_1), t_s(u_2), ..., t_s(u_n)\}$ to minimize the energy consumption while ensuring the performance degradation of each transmission task u to be within the user expectation bound $\tilde{\Phi}(u)$.

For each task u, we need to guarantee

$$\phi_u(t_e(u) - t_a(u)) \leq \tilde{\Phi}(u). \quad (8)$$

Based on (5) and (6), the total energy consumption estimation can be calculated as

$$E(U, t_s(U), \Gamma) = \widetilde{E}_d(U, t_s(U), \Gamma) + \widetilde{E}_t(U, t_s(U), \Gamma). \quad (9)$$

Therefore, the optimization problem can be formulated as

$$\min E(U, t_s(U), \Gamma) \quad (10)$$
$$\text{subject to Constraints (1), and (8).}$$

Constraints (8) can be translated as $t_e(u) \leq t_{dd}(u)$, where $t_{dd}(u) = t_a(u) + \phi_u^{-1}(\tilde{\Phi}(u))$ denotes the upper bound of $t_s(u) + \mathcal{D}_t(u)$. This bound takes into account the application profile ϕ_u and the user requirement $\tilde{\Phi}(u)$ for a task u.

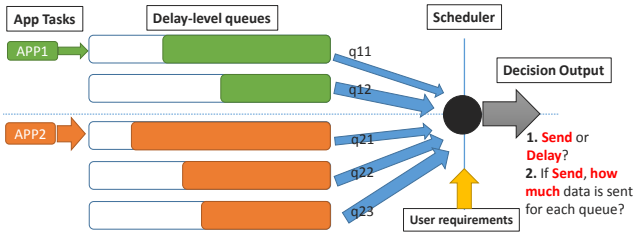


Fig. 3: Example of PerES scheduler design

Theorem 1. The performance-aware energy optimization problem (10) is *NP*-hard.

Proof: See the detailed proof in our technical report [30]. □

4 Scheduling Design

4.1 Overview

Since the optimization problem is NP-hard, in this section, we design an online scheduler **PerES** (Performance-aware Energy Scheduler) that could easily run in the practical system. In the example shown in Fig. 3, at the beginning, the application tasks are added into several queues. The PerES scheduler will make the transmission decisions in each slot based on the queue status and user requirements. The decisions will determine if the data should be sent, and if yes, how much data to send from each queue.

There are two key differences between the designs of ours and others. First, we use multiple queues for each application to explicitly control the task delay. Since the Lyapunov theory is primarily applied for stabilizing the queue, following the standard Lyapunov optimization model, most existing work employs the conventional FIFO queue and represent the delay performance with the queue length [19]. In reality, the queue length of a conventional FIFO cannot always capture the delay performance. For example, when no new tasks are added into a FIFO queue for a long duration, the queue length does not change but the task delay will be big. In this work, we propose a novel queue management method named "delay-level queue" to make it possible to capture the task delay with the queue length all the time.

Second, we take the user requirements as the performance constraints on multiple applications. In the example of Section 3.1, we have shown that the conventional FIFO queue is inefficient to deal with different delay requirements from multiple applications. In this work, after capturing the delay, we also need to know how to schedule transmissions from the proposed "delay-level queues" with respect to the user requirements.

The problem can be solved with four steps as shown in Algorithm 1. We first relax the problem from meeting the performance bound of each application task to meeting the bound of total task performance (section 4.2). To solve the relaxed problem, we construct the delay-level queues to handle the tasks from different applications (section 4.3). Next, based on the current network status, we allocate different data rates to each delay-level queue (section 4.4). Furthermore, we adjust the parameters so the algorithm can

Algorithm 1 PerES: Performance-aware Energy Scheduler

1: Obtain the relaxed problem by meeting the bound of total task performance cost
2: Construct the delay-level queues to handle the task delay in different application groups
3: Allocate different rates to each delay-level queue
4: Adaptively adjust the parameters with respect to user performance requirements

adaptively converge to the optimal energy bound subject to user performance requirements (section 4.5).

4.2 Problem Relaxation

In order to reduce the tail energy associated with the RRC process, an online scheduler can send the traffic in batch. However, as the mobile traffic and wireless link quality can not be accurately predicted, this process will inevitably introduce some delay in transmission and may even lead to violation of some given performance bounds, e.g., the task deadline.

Therefore, it is not possible and practical for an online scheduler to always satisfy the performance bound for each task in real-time. Instead, we relax the constraint from meeting each task's performance cost bound to meeting the bound of total task performance cost denoted as Ω. A higher performance bound is due to a longer tolerable delay weighted by application profiles and user preference. This would motivate the scheduler to aggregate the less-urgent (or less-interested to the user) traffic for lower tail energy and wait for the better channel condition to transmit.

In our relaxed scheduling problem, the scheduler requires no future information of the traffic, but makes decision in each time slot to obtain the long-term benefits, i.e., optimizing both the energy and performance in a long enough time scale ($\Gamma \to \infty$). We derive the practical optimization model as follows.

Define $PW(\tau)$ and $PD(\tau)$ as the energy consumed and performance cost in time slot τ with their average values $\overline{PW}(\tau)$ and $\overline{PD}(\tau)$ respectively, we have

$$\overline{PW}(\Gamma) = \frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1} PW(\tau) = \frac{1}{\Gamma}E(U, t_s(U), \Gamma) \tag{11}$$

$$\overline{PD}(\Gamma) = \frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1} PD(\tau) = \frac{1}{\Gamma}\sum_{u\in U}\phi_u(\mathcal{D}(u)) \tag{12}$$

where Γ is the scheduling period. Then the following performance condition should be satisfied:

$$\overline{PD}(\Gamma) \le \Omega \tag{13}$$

The online optimization problem is formulated as:

When $\Gamma \to \infty$,

$$\begin{aligned} &\text{Minimize } \overline{PW}(\Gamma) \\ &\text{subject to Constraints (1) and (13).} \end{aligned} \tag{14}$$

For clarity, we list the related notations in Table 3.

4.3 Delay-level Queue Construction

To take into account the user performance in our scheduling, we classify the applications into groups and queues based on the user preferences and application profiles. Then we need to explicitly control the delay performance for the tasks in each queue. In the following, we present a novel *delay-level queue* to address this problem.

4.3.1 Group Classify

We first classify applications into n groups $G = \{G_1, G_2, ..., G_n\}$ based on two properties: the form of *profile function* (decided by application profiles) and the preference *weight* (decided by user preference). Two tasks belong to the same group if they share the same properties. Let $F_i(\mathcal{D}(u))$ denote the performance cost per byte of the task u with a delay $\mathcal{D}(u)$ from group i, i.e., $F_i(\mathcal{D}(u)) = \phi_u(\mathcal{D}(u))/\mathcal{S}(u)$. Generally, the same type of applications are classified into one group (e.g., Email applications). In the extreme case, each application can have its own group.

4.3.2 Queue Update

Generally, the delay sensitivity of a task is reflected by its deadline. Tasks in one group have the same deadline and can be inserted into one of m queues (called the **delay-level queues**) based on their current delay. A queue q_{ij} corresponds to one *delay level* j from group i. The delay-level interval is defined as d_i/θ where d_i denotes the deadline of tasks from group i, and θ is the delay granularity of dividing the group into queues. Note that m is large enough so that $m \times d_i/\theta$ is the maximum waiting delay that users can tolerate for any task in the queue.

The scheduler will update the delay-level queues in each time slot. If the delay value of a task u from group i is between $d_i/\theta \times (j-1)$ and $d_i/\theta \times j$, it will be put in q_{ij}; when its delay exceeds $d_i/\theta \times j$, it will be popped out of q_{ij} and be pushed into the next delay-level queue $q_{i(j+1)}$. This update operation starts from the tasks with the highest delay to the ones with the lowest delay. Hence after a linear traversal update from the head task to the end task in each queue, the ordered property of delay still holds for all the tasks.

The scheduler takes the delay level as the reference to determine the transmission sequence and time, i.e., the delay of tasks in q_{ij} is taken as $d_i/\theta \times j$. Define the length of the minimum time unit as ℓ_t. If the delay-level interval d_i/θ equals ℓ_t, $d_i/\theta \times j$ is equivalent to the accurate queuing delay. Let φ_{ij} denote the performance cost per byte of queue q_{ij}, i.e., $\varphi_{ij} \triangleq F(\mathcal{D}(u))$ where u is a task in queue q_{ij}.

The advantage of above queue management is that the length of each delay-level queue can effectively reflect the

TABLE 3: Basic Notations for Online Model

Notations	Definitions
$c(t)$	bandwidth capacity at slot t
$PW(t)$	energy cost at slot t
$PD(t)$	performance cost at slot t
Ω	the threshold of maximum performance cost
q_{ij}	queue of *delay level* j at group i
φ_{ij}	performance cost per byte of queue q_{ij}
V	the trade-off parameter between energy and performance
$\gamma_{ij}(t)$	average transmission task arriving rate for queue q_{ij}
$r_{ij}(t)$	average transmission rate allocated for queue q_{ij}

delay impact. Then we are able to generate the following theorem to directly link the delay-level queues in each group with the user performance cost caused by queueing tasks.

Theorem 2. Denote $PD_{ij}(t)$ as the total user performance cost of the tasks in queue q_{ij} in time slot t. Denote $\gamma_{ij}(t)$ as the average transmission task arriving rate for queue q_{ij} and $r_{ij}(t)$ the average transmission rate allocated for the queue, we have:

$$PD_i(t+1) = PD_i(t) - \sum_{j=1}^{m} \varphi_{ij} r_{ij}(t) + \varphi_{i1}\gamma_i(t) \quad (15)$$

where $PD_i(t) \triangleq \sum_{j=1}^{m} PD_{ij}(t)$ and $\gamma_i(t) \triangleq \sum_{j=1}^{m} \gamma_{ij}(t)$. Moreover, for a task u in queue q_{ij}, if u is transferred at the current slot, we have the task delay $\mathcal{D}(u)$ counted as $d_i/\theta \times j + 1$, i.e., $\varphi_{ij} = F(d_i/\theta \times j + 1)$.

Proof: See the detailed proof in our technical report [30]. □

4.4 Queue Scheduling

After the queue construction, there are two remaining things to do: obtaining the user requirement input and allocating data rates for the delay-level queues.

4.4.1 User Requirement Input

Our online scheduler runs as a traffic manager application and uses a UI for the input of user requirements. It lists all the application icons and offers the setting of the user preference weight and maximum tolerable delay for each selected application. Thus users can easily change his preferences on the UI. The task deadlines and application profile functions (i.e., the application sensitivity to delay) are provided from application developers by default (through calling our SDK) to benefit from the energy savings of our solution. For ease of use, the maximum tolerable delay can be easily set as a sliding bar so it can be tuned to a value from 0% to 100% of the deadline to be satisfied. With different application deadlines, the same fraction refers to different tolerable delay. The traffic manager will update the average data arriving rate for each application based on history traffic statistics. Hence we finally obtain the user preference weight $\widehat{w}_i$, maximum tolerable delay $\widehat{d}_i$ and the average data arriving rate $\widehat{\gamma}_i$ on each application i. Based on the performance cost metric defined in equation (2), the user-defined time-average performance cost bound Ω is computed as $\sum_{i=1}^{N} \widehat{w}_i \times f_p(\widehat{d}_i) \times \widehat{\gamma}_i$, where N is the number of selected applications.

4.4.2 Rate Allocation

In this part, we are able to derive our rate allocation algorithm (called **RAA**) in *PerES* to decide the data rate for each delay-level queue. Following the Lyapunov framework [31], our Lyapunov function is defined as:

$$L(t) \triangleq \frac{1}{2}\sum_{i=1}^{n} (PD_i(t))^2 \quad (16)$$

Denote the vector $\overrightarrow{PD}(t) \triangleq \{PD_{ij}(t)|_{1\le i\le n, 1\le j\le m}\}$ as the performance cost of each queue at time t. The one-step Lyapunov drift $\Delta(t)$ is defined as:

$$\Delta(t) \triangleq \mathbb{E}\{L(t+1) - L(t)|\overrightarrow{PD}(t)\} \quad (17)$$

Algorithm 2 RAA: Rate Allocation Algorithm

Input: V, t // the current time slot
Output: $\mathbb{R}_m$ // the data rate allocation set

```
Update the delay-level queues by (15)
for each queue q_ij do
    compute A_ij by (22)
end for
R_t ← R_sig(t)
Sort q_ij in the descending order of A_ij as Q
while R_t > 0 and Q ≠ φ do
    Pop the q_ij in Q who has the largest A_ij
    r_ij ← min{Size(q_ij), R_t}
    R_t ← R_t − r_ij
end while
Get the rate allocation set ℝ_m = {r_ij} and compute the objective D_m in (19)
if D_m ≤ 0 then
    Set all the elements in ℝ_m to 0
end if
return ℝ_m
```

We add the energy minimization objective into the Lyapunov drift by the *drift-plus-penalty* form $\Delta(t) + V\mathbb{E}\{PW(t)|\overrightarrow{PD}(t)\}$ and obtain the following lemma:

Lemma 1. Assume that the data arrival process $\widetilde{\lambda}(t)$, and the transmission process $\widetilde{u}(t)$ have finite expectation, i.e., $\exists$ constants $\mathcal{A}$ and $\mathcal{U}$ such that $\mathbb{E}\{\widetilde{\lambda}(t)\} < \mathcal{A}$ and $\mathbb{E}\{\widetilde{u}(t)\} < \mathcal{U}$. We have

$$\begin{aligned} &\Delta(t) + V\mathbb{E}\{PW(t)|\overrightarrow{PD}(t)\} \\ &\le B - \mathbb{E}\Big\{\sum_{i=1}^{n}\big[PD_i(t)\mathbb{E}\{\sum_{j=1}^{m}\varphi_{ij}r_{ij}(t)|\overrightarrow{PD}(t)\}\big] \\ &\quad - V\cdot PW(t)|\overrightarrow{PD}(t)\Big\} + \sum_{i=1}^{n}[PD_i(t)\cdot\varphi_{i1}\gamma_i] \end{aligned} \quad (18)$$

where $B \triangleq \frac{1}{2}\{(\sum_{i=1}^{n}\sum_{j=1}^{m}\varphi_{ij}^2)\cdot\mathcal{U}^2 + (\sum_{i=1}^{n}\varphi_{i1}^2)\cdot\mathcal{A}^2\}$ and φ_{ij} is a positive constant.

By applying the dynamics of queue performance (15) into the Lyapunov drift (17), we can obtain the fact of (18). See the detailed proof in our technical report [30]. Based on the Lyapunov design principle, the optimal scheduling decision is to minimize the *drift-plus-penalty* expression in each time slot. *PerES* minimizes the RHS of (18) to guarantee the performance stability with the minimal power consumption:

$$\begin{aligned} \text{Maximize } D(t) = &\sum_{i=1}^{n}\Big[PD_i(t)\times\sum_{j=1}^{m}\{\varphi_{ij}\times r_{ij}(t)\}\Big] \\ &- V\times PW(t) \quad (19) \\ s.t. \quad &\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}(t) \le c(t) \quad (20) \\ &0 \le r_{ij}(t) \le \mathcal{S}(q_{ij}) \quad (21) \end{aligned}$$

where $c(t)$ is the bandwidth of the wireless link at time t and $\mathcal{S}(q_{ij})$ is the total data size of tasks in the queue q_{ij}. One key difference between our model and other existing Lyapunov models (e.g., [19], [20]) is that the rate allocation constraint (20) models the bandwidth competition among the delay-level queues of different applications, which is important

Algorithm 3 SVA: Self-adaptive V Algorithm

Input: Ω, t //the current time slot
Output: Real-time Scheduling Decision
1: **if** t equals 0 **then**
2: $V(t) = 0$
3: **else**
4: calculate $\overline{PD}(t)$ by (12)
5: **if** $\overline{PD}(t) < \Omega$ **then**
6: $V(t) = V(t-1) + \delta$
7: **else**
8: $V(t) = V(t-1)/2$
9: **end if**
10: **end if**

when considering the differences in application profiles and user requirements on applications.

As Algorithm 2 illustrates, *RAA* first updates the performance degradation $PD_i(t)$ of all the queues by (15) (line 1). *RAA* achieves the optimal rate allocation set $\mathbb{R}_m = \{r_{ij} | i = 1, 2, 3, ..., n; j = 1, 2, 3, ..., m.\}$ by solving the linear programming problem (19) with the following procedure. First, based on Theorem 2, the weight A_{ij} of each queue is computed to capture the impact of delay on user performance (line 2-4):

$$A_{ij} = PD_i(t) \times \varphi_{ij} = PD_i(t) \times F_i(d_i/\theta \times j + 1) \quad (22)$$

Next, the total bandwidth $c(t) = R_{sig}(t)$ is allocated to the queues in the descending order of A_{ij} until it is completely allocated as lines 5-11 in Algorithm 2.

After rate allocation for each delay-level queue, *RAA* computes the objective value D_m by applying the rate allocation set $\mathbb{R}_m$ into the objective function of (19). For a quick computation for this control decision, according to (11), $PW(t)$ can be computed as:

$$PW(t) = P_{sig}(t) \times \{\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}(t)/c(t)\} + E_{Tail}(\Delta t') \quad (23)$$

where $\Delta t' = t - t_{last}$ and t_{last} denotes the last time slot that transfers data. If $D_m \leq 0$, the rate allocation set $\mathbb{R}_m$ is set to all zeros.

In Algorithm 2, lines 1-4 perform the update of the delay-level queues with cost $O(k)$, and then compute A_{ij} for each q_{ij} by $O(mn)$ operations. Line 6 sorts q_{ij} by $O(mn\log(mn))$ operations, and it then takes $O(mn)$ operations to allocate the rate for each q_{ij} in lines 7-15. Hence the total time complexity of *RAA* (Algorithm 2) is $O(mn\log(mn) + k)$, where n is the number of application groups, m is the number of delay-level queues per group and k is the number of tasks in buffer.

4.5 Bound Analysis and Approaching

In this section, we first give the boundness analysis of *PerES* and then present an efficient method which adapts the parameters to efficiently approach the bound.

4.5.1 Bound Analysis

The properties behind the scheduling decision (19) imply the following theorem:

Theorem 3. Assume that the data arrival rate is strictly within the network capacity region, and the online scheduling decision (19) is applied by *PerES* at each time slot. For any control parameter V of *PerES* where $V > 0$, it generates the time-average power consumption $\overline{PW}_\infty$ and time-average performance cost $\overline{PD}_\infty$ satisfying that:

$$\overline{PW}_\infty = \lim_{\Gamma\to\infty}\sup\frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{PW(\tau)\} \leq P^* + \frac{B}{V} \quad (24)$$

$$\overline{PD}_\infty = \lim_{\Gamma\to\infty}\sup\frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{PD(\tau)\} \leq \frac{B + VP^*}{\varepsilon} \quad (25)$$

where B and ε are positive constants. P^* is the theoretical optimal time-average power consumption.

Based on Lemma 1 and the similar method for deriving Lyapunov bound in [19], [31], we could obtain the fact of (24) and (25). See the detailed proof in our technical report [30].

4.5.2 Adaptive Scheme

The parameter V in (19) is the key tradeoff parameter between the energy cost and the performance cost. In Theorem 3, following the Lypapunov theory, a fixed V setting will make the power cost and the performance cost converge to a stable value close to their corresponding upper-bound. For example, based on equation (24), a *larger* V value will make the energy cost converge to a *smaller* stable value, while this will make the stable value of the performance cost *larger* based on equation (25). Hence, it is good to use a large enough V value to reduce the energy cost while constraining the performance cost to be within Ω.

Inspired by the above analysis, we could solve the online optimization problem (14) in the following way. For any given user requirement Ω, if we could find a large-enough V value that makes the system work with a performance cost close to but within Ω, then according to (24), the upper bound of the power consumption is minimized to its optimal value. However, tuning such a proper V value is a long-standing problem still without known efficient solutions. The traditional way of solving this convergency problem falls into the region of calculating a *magic number* of V based on some heuristic information, such as the method proposed in [19], or assuming a *nice range* of V that is possible to apply [20]. However, it is difficult to analyze the performance and hard to apply these conventional schemes in a complex mobile environment.

We design a dynamic scheme, called **SVA** (Self-adaptive V Algorithm) to handle this tough issue in a practical and fast way. Rather than figuring out a *good* V value before the scheduling, we search for a proper value during the scheduling. Starting from V=0, we increase the V value while the performance cost is within the required Ω, and reduce the V value when Ω is exceeded to constrain the performance cost and satisfy the user requirement. As Algorithm 3 shows, the *SVA* monitors the current time-average performance cost value $\overline{PD}$. When it is lower than the user requirement, *SVA* increases the V linearly; otherwise, *SVA* cuts the V value down to a half. The intuition behind *SVA* comes from the congestion avoidance scheme utilized by the TCP protocol.

The AIMD (Additive Increase Multiplicative Decrease) is primarily designed for the TCP window to increase to a value that gets the system work close to but within the congestion bound, thus the throughput is maximized. Our experiment validates that using the *AIMD* scheme in our system could also enable a fast convergency to the required performance bound, and thus the time-average power consumption is minimized to its optimal bound.

To summarize, our *PerES* makes the scheduling decision in each slot. It determines the control parameter V with respect to the user performance requirement Ω (Algorithm 3), and then calls *RAA* (Algorithm 2) to set the rate allocation $\mathbb{R}_m$. *PerES* serves all the queues by $\mathbb{R}_m$ for data transferring. If all the elements in $\mathbb{R}_m$ equal zero, *PerES* keeps all the queues waiting.

4.6 Offline Solutions as Benchmark

To provide a benchmark for the evaluation of the online algorithm, we present an offline algorithm for an approximation solution of the original NP-hard optimization problem (Theorem 1).

Consider the offline scheduler as a black box, which receives n tasks in their arriving order, and outputs the tasks in another order following the schedule. Next we will present an algorithm which can find the local optimal solution for any fixed output order ω. Intuitively, we can find the global optimal solution by traversing all the local optimal solutions under the $n!$ possible output orders. A fixed output order ω can ensure the optimal substructure of the *DP* (Dynamic Programming) solution. This allows us to design a *DP* algorithm, *Loc-TRES* (Local Transimission Energy Scheduler) to achieve a *local optimal* solution offline.

To begin with, we sort $u \in U$ by order ω. Let g_i denote the optimal objective value for scheduling the first i tasks, while $f_{i,k}$ denotes the optimal objective value of the first i tasks if u_i is scheduled to transmit in time slot k. Let Π_i define the valid range of $t_s(u_i)$, i.e, $\Pi_i = \{t_s(u_i)|t_s(u_i) \geq t_a(u_i) \text{ and } t_e(u_i) \leq t_{dd}(u_i)\}$. Then we can build the dynamic programming equations as $f_{i,k} = \min_{j\in\Pi_{i-1}}\{E_{trans}(k, k+\mathcal{D}_r(u_i)) + E_{tail}(k-(j+\mathcal{D}_r(u_{i-1}))) + f_{i-1,j}\}$ and $g_i = \min_{k\in\Pi_i}\{f_{i,k}\}$. Based on the iterative equations on $f_{i,k}$ and g_i, we can easily obtain g_n for the final schedule. Hence the time complexity of Loc-TRES is $O(nR^2)$, where R=$\max_{1\leq i\leq n}\{t_{dd}(u_i) - t_a(u_i)\}$.

To give another evaluation benchmark, we also develop a brute-force-based offline scheme (denoted as Opt-TRES) to search the global optimal solution, which uses the input offline information including all the signal traces and application task traces. Although its worst-case complexity is exponential in time due to the NP-hardness, we speed it up in most cases by using a branch-and-bound search based on Loc-TRES. Specifically, we employ the solution found by *Loc-TRES* following the task arriving order to initialize the optimal solution, which will contribute to the fast boundary pruning operation during the search. It serves as the optimal offline scheduler and is compared with both Loc-TRES and our online algorithm in Section 5.

TABLE 4: Application Settings

App ID	1	2	3	4	5
Deadline (s)	10	200	400	800	1600
Weight	1/10	1/200	1/400	1/800	1/1600

5 Evaluations

5.1 Evaluation Setup and Methodologies

We implement *PerES* as a traffic management application on a smartphone (Google Nexus S). It utilizes IPTABLES (an existing system tool in Android) to redirect the data flow of each application to a specific interface of our unified scheduler so the transmission tasks can be buffered for a specific time duration on the phone. All the algorithms compared in this work are implemented inside the traffic management application on the phone and can be selected to run based on the UI. All the simulations in this paper are conducted on the smartphone. We monitor the signal strength of the cellular network interface and measure the energy consumption and performance metrics. To acquire the transmission rate and power under different signal strength, we take the phone to record 3G signal traces in 20 different places of Tsinghua University. The power value is measured by the Monsoon Power Monitor device, and we find that the power P_{sig} (mW) and the data rate R_{sig} (kBps) could be fitted as a linear function with the signal strength [14], [16].

In the first part, to facilitate the study on performance insights without loss of generality, we simulate the changing signal as a sine function in the range of -50dbm to -110dbm with a random interference between -10dBm and 10dBm. The Rayleigh fading model is also applied to simulate the dynamics in wireless channels. In the later part, we further apply real signal traces to validate the energy efficiency. The task arrives following the *Poisson Distribution*. In reality, an application's task is expected to be transmitted before the arrival of its next task, hence we set the average arrival interval of the tasks as their waiting *deadline*. The data size of one task is set as a random variable in $(0, 500]$ Kbytes. The linear coefficient δ of *SVA* in *PerES* is set to 0.001, and the detailed application settings are shown in Table 4.

We first evaluate the performance of the online algorithm *PerES* against offline algorithms in different scenarios over Γ=10000 time slots (one sec as one slot). The offline solutions are obtained by *Loc-TRES* and *Opt-TRES* to serve as a performance reference. *Loc-TRES* sets the arriving order of tasks as its initialized order ω, while *PerES* is evaluated under different levels of performance loss. *PerES-X* scheduler extends from *PerES* by allowing for a deadline violation ratio X. For the offline algorithms, the performance degradation bound $\tilde{\Phi}(u)$ of each task is set to $\phi_u(t_d(u) - t_a(u))$ to make sure that $t_{dd}(u) = t_d(u)$. It is equivalent to finding the optimal schedule by constraining the delay of each task to be within its deadline. In this setting, the offline algorithms ensure that there is no deadline violation. For the online scheduler *PerES*, the user-defined performance bound Ω is varied to get different levels of deadline violation ratios. The default scheme *Immediately* is also included as a reference, i.e., it transfers the data immediately upon the arrival of tasks.

Next, four non-predication based online schedulers are compared in details, i.e., *TailEnder* [10], *SALSA* [19], *eTime*

[20] and *PerES*. Table 5 shows their different features. They represent the consideration for signal strength impact, different granularity of tail energy counting (e.g., "Coarse" means only one average tail power is counted), deadline awareness, application profiles, user preference and convergence scheme (e.g., "Static" means the trade-off parameter between energy and performance is set as a constant value) separately. TailEnder is a const-setting-based online scheduler while the others are designed under the Lyapunov framework. We develop the *E-P panel* to compare the scheduling optimality for energy consumption E (the sum of data transmission energy and tail energy) and performance degradation P (i.e., $\overline{PD}(\Gamma)$). In the E-P panel, each set of points for schedulers is acquired by linearly increasing the parameter V of *SALSA*, *eTime* and the Ω of *PerES* in equal pace, and there is no parameter change in *TailEnder*.

We evaluate the performance of online schedulers through simulations over a period of 10000 time slots. We analyze the impact of various parameters listed in Table 6. During the simulation, when one factor is changed, other factors are set to their default values. Each data point recorded in our simulation results is the average value over 20 random problem instances. We evaluate four representative profile functions. All functions satisfy Property 1, but have different forms before and after the deadline. Specially, we define two typical changes before and after the deadline: unchanged (U) or linearly increasing (L), i.e., we have four forms of the profile function: UU, UL, LU, LL, which corresponds to real-world applications. For example, function UL captures the feature of the email applications that will not trigger the users' concerns until certain user tolerance time is violated, while a further delay will cause worse experience.

For comparative analysis, we evaluate the tail energy and data transmission energy for energy metrics, while the deadline-violation ratio and normalized average delay for performance metrics. The *deadline violation ratio* denotes the size of tasks whose delay exceeds their deadline divided by the total data size. The *normalized average delay* $\overline{D}$ denotes the sum of weighted (normalized preference weight) delay of all tasks divided by the total data size of all tasks: $\overline{D} = \frac{\sum_u \{w_u \times \mathcal{D}(u) \times \mathcal{S}(u)\}}{\sum_u \mathcal{S}(u)}$.

5.2 Optimality Analysis

As Fig. 4 shows, the total energy consumption and energy saving of different schedulers increase approximately linearly as the simulation time increases. *Opt-TRES* can get on average 38% energy saving compared to the default strategy. Compared with *Opt-TRES*, the energy consumption of *Loc-TRES* is within a factor 1.06 of the optimal solution, which demonstrates its high approximation ratio. It also indicates that the arriving order works well as the initialized order for *Loc-TRES* in real applications. Both offline algorithms *Opt-TRES* and *Loc-TRES* intend to minimize energy while not violating task deadlines.

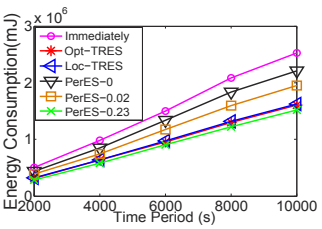


Fig. 4: Optimality Comparison for Loc-TRES and PerES

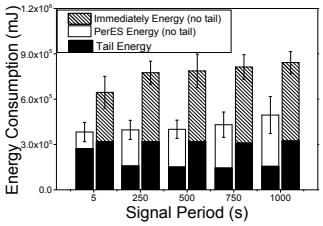


Fig. 5: Energy Performance over Different Signal Periods

PerES-0 is the online scheduler that adjusts the performance bound Ω of *PerES* to meet the same constraint that zero deadline violation is satisfied. On average, *PerES-0* is seen to achieve 13% energy saving compared to the default strategy without deadline violation. Compared with *Opt-TRES*, it is within a factor 1.4 of the optimal total energy consumption and within a factor 1.1 of the optimal energy saving. If mobile users can tolerate more performance loss, *PerES* could obtain much higher energy efficiency. When deadline violation ratio is 0.02, *PerES-0.02* achieves twice energy saving that of *PerES-0*. When the deadline violation ratio is 0.23, *PerES-0.23* can achieve performance comparable to *Opt-TRES*. It even achieves a little more energy saving than *Opt-TRES* at the cost of some deadline violation.

5.3 Parameter Analysis

5.3.1 Impact of Signal Period

As Fig. 5 shows, we test the energy performance of *PerES* and *Immediately* over different variation periods of the signal strength. We can see that the total energy consumption of both *PerES* and *Immediately* increase with the signal variation period. When the period is smaller, there are more opportunities to use good signal points for transmitting data, which helps reduce the data transmission energy. For a small variation period as 5 seconds, the tail energy of *PerES* is the highest among all periods studied while that of *Immediately* changes little. This is because when the period is comparable to the length of tail time, there are more good signal points for *PerES* to transfer data with small intervals, which incurs an additional high tail energy between two neighboring good signal points. *Immediately* does not react to the change of signal and thus has similar tail energy performance.

5.3.2 Impact of Deadline Range and Dynamic V

As a default setting, our scheduler is designed for delay-tolerant applications generally with deadlines in the unit of minutes, such as uploading video clips [19] or the periodic background updates like the news and emails applications [24], [32]. The work in [32] shows that a typical deadline set

TABLE 5: Evaluation Schedulers

Scheduler	Signal	Tail	Deadline	Profile	Weight	Converg.
TailEnder	-	Coarse	✓	-	-	-
SALSA	✓	-	-	-	-	Static
eTime	✓	Coarse	-	-	-	Static
PerES	✓	Fine	✓	✓	✓	Dynamic

TABLE 6: Evaluation Parameters Setup

	Default	Range
Minimum Arrival Interval (s)	10	$1 \sim 100$
Minimum Deadline (s)	10	$1 \sim 100$
Maximum Preference Weight	1/10	$1/100 \sim 1$
Signal Variation Period (s)	500	$5 \sim 1000$
Average Signal Strength (dBm)	-80	$-110 \sim -50$
Profile Function	LL	$\{UU, UL, LU, LL\}$
Delay Granularity (θ)	10	$1 \sim 100$

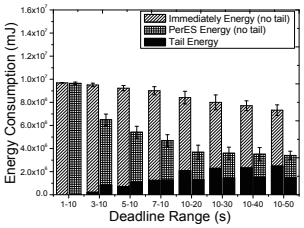


Fig. 6: Performance over different deadline ranges

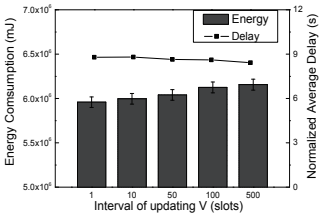


Fig. 7: Performance over different V update intervals

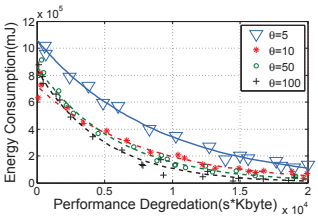


Fig. 8: Delay-granularity Accuracy Impact

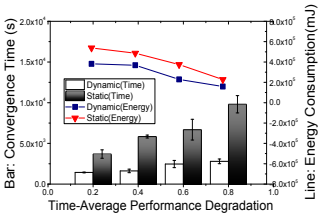
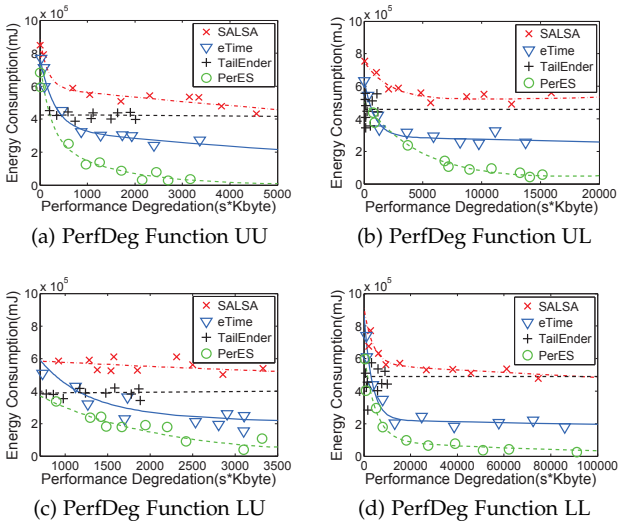


Fig. 9: Convergence: Dynamic vs. Static Strategy



(a) PerfDeg Function UU (b) PerfDeg Function UL

(c) PerfDeg Function LU (d) PerfDeg Function LL

Fig. 10: The E-P Panels for Different Profile Functions

on smartphone is 1 minute. However, to make a complete comparison, in this section we also test the performance of our scheduler when deadlines are generally much smaller than 1 minute. As Fig. 6 shows, we evaluate the performance of Immediately and PerES with different deadline ranges for all the tested applications. We can see that with the smallest deadline larger than 3 seconds, the energy savings of PerES compared to Immediately is about 28% to 54%.

However, for the *extreme* setting with the smallest deadline as 1 second, PerES has little energy savings over Immediately. Obviously, such a small delay tolerance provides little scheduling opportunity to optimize the energy. It is impossible to delay the tasks whose deadlines are only 1 second (which can be considered as real-time tasks), as our scheduler runs once per second and any additional delay would violate its deadline. Since we set the tasks to arrive with their average intervals as deadlines, the ones with the smallest deadline contribute to most of the traffic and thus create the final energy impact. Moreover, we can see that the tail energy increases when the smallest deadline increases, which is contributed by longer idle intervals between two successive arrival tasks. The tail energy of PerES is about 37% lower than Immediatelly when the deadline range becomes larger because PerES gets more opportunity to batch the packets when the tasks are more delay-tolerant.

Rather than updating the value of V in each time slot as in Algorithm 3, we studied the performance of *PerES* over different V update intervals in Fig. 7. When the update interval of V parameter is less than 10 slots, the performance of energy and delay is similar. The energy consumption increases gradually when the update interval is further increased. This indicates that we can reduce the computation overhead by updating the V value every 10 slots with little performance degradation.

5.3.3 Delay-granularity Setup

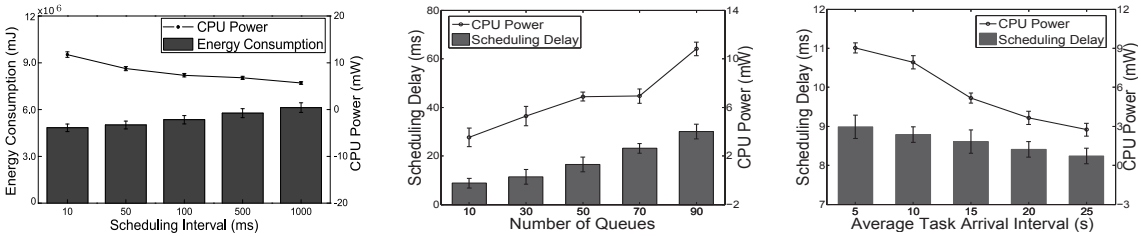
As Fig. 8 shows, we test four levels of accuracy setting for the *E-P panel*. As θ increases, the delay granularity for scheduling decreases. We can find that, from $\theta = 5$ to 10, the performance has a big improvement, and the energy consumption decreases faster when the performance degradation is set as $\theta = 10$. When the granularity further decreases, i.e., $\theta = \{10, 50, 100\}$, the performance does not have significant change. The reason is that the smallest deadline of the tested applications is 10 seconds, which reaches the finest granularity of its delay levels as the minimum division granularity is one second (i.e., one time slot). Since in this study the average task arrival interval is the same as the deadline, the tasks with the smallest deadline arrive more frequently than other application tasks, and thus generate more traffic and have the main effect on the performance. In the following, we set $\theta = 10$ to achieve the best balance between the overhead of queue management and the good performance of scheduling.

5.3.4 Dynamic vs. Static Schemes

As Fig. 9 shows, we set four levels of time-average performance requirement Ω between 0.2 and 0.8. Existing strategies normally pre-compute a static V value by some heuristic method [19] or obtain empirical values in experiments [20]. In our study, for a given Ω setting, the static strategy uses the optimal static V value obtained by adjusting the V value in the experiment to make the time-average performance degradation converge to Ω exactly, and keeps the V value as a constant during the scheduling. We find that *SVA* (dynamic strategy) converges to the performance target 2-5 times faster than that of the static strategy. We can see that larger performance requirement Ω implies larger room for energy optimization. Furthermore, for the same performance requirement, a faster convergence speed of *SVA* results in higher energy efficiency than the static strategy during a specified scheduling period. Therefore, our scheme can react fast to the change of user requirements.

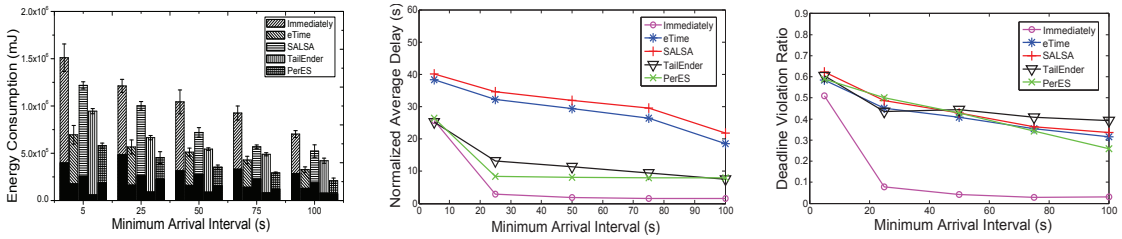
5.3.5 Profile Functions

As Fig. 10 shows, we evaluate schedulers in *E-P panels* with four different profile functions. We could observe the obvi-



(a) Scheduling Interval (b) Queue Number per Group (c) Average Task Arrival Interval

Fig. 11: Scheduling Overhead on CPU Power and Delay



(a) Energy Consumption (b) Normalized Average Delay (c) Deadline Violation Ratio

Fig. 12: Performance over Different Arrival Interval

ous tradeoffs between the energy consumption and performance degradation for all other schemes except TailEnder, where the faster dropping of energy with the degradation of performance indicates a better performance. *eTime* performs better than *SALSA* by embedding the tail energy into its control decision. Our *PerES* performs the best for all the profile functions tested. On average its energy efficiency is 2 times that of the *eTime* and *TailEnder*, and 4 times that of the *SALSA* for a given performance. TailEnder's results are constrained within a small range, as it schedules the transmission mostly around the deadline. This helps to improve its performance but limits the room for its energy optimization.

5.3.6 Scheduling Overhead

While saving energy for data transmissions, it is also important for the scheduler itself to introduce low overhead for computation and batch scheduling. We measure the average CPU power consumption and scheduling delay of running our *PerES* scheduler application. To exclude the power consumed by data transmission, we run the experiments by not actually transmitting the data but only executing exactly the same algorithm procedures as if the data were transferred. We further subtract the system base power of not running any application to obtain the pure scheduling overhead on CPU power. The scheduling delay is measured by recording the time stamps in each round of scheduling.

We first show the impacts of scheduling frequency on energy in Fig. 11a. By decreasing the scheduling interval from 1000ms to 10ms, the average CPU power on computation is doubled while the total energy consumption is decreased by 23%. Obviously, a more frequent scheduling would increase the computation overhead, but the real-time and fine-gained task handling also contributes to higher transmission energy efficiency. However, a frequent scheduling may keep CPU busy which would degrade the performance of other applications on the device. Considering the tradeoff between energy and performance, the scheduling interval is suggested to be between 100ms and 1000ms.

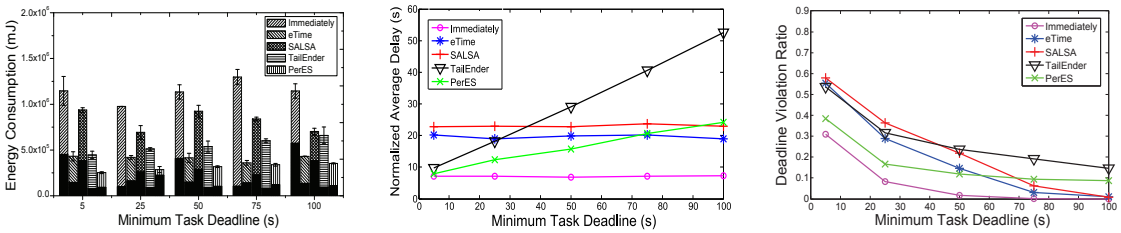
We further analyze the scheduling overhead on both power and delay under various network loads. As Fig. 11b shows, both the scheduling power and delay increase with the number of queues. In Fig. 11c, we vary the average task arrival interval to change the number of tasks to handle in buffer. As expected, the scheduling power and delay decrease when the task arrival interval increases. As our previous analysis illustrates, the computation overhead of *PerES* is $O(mn\log(mn)+k)$ where mn is the total number of queues and k is the number of tasks in buffer, which matches the results in Fig. 11. In both cases, the average CPU power is around 10mW, which is an order of magnitude lower than both the system base power (160mW) and the data transmission power (730mW) (see Fig. 11b and Fig. 11c). Moreover, the average scheduling delay per round is within 30ms, which is sufficiently low to run our scheduler per slot (i.e., one second in this study).

5.4 Comparative Analysis

We compare the performance metrics for different schedulers listed in Table 5. The black part of the energy metric represents the tail energy part while the non-black part represents the data transmission energy part in Fig. 12a, Fig. 13a, Fig. 14a and Fig. 15a.

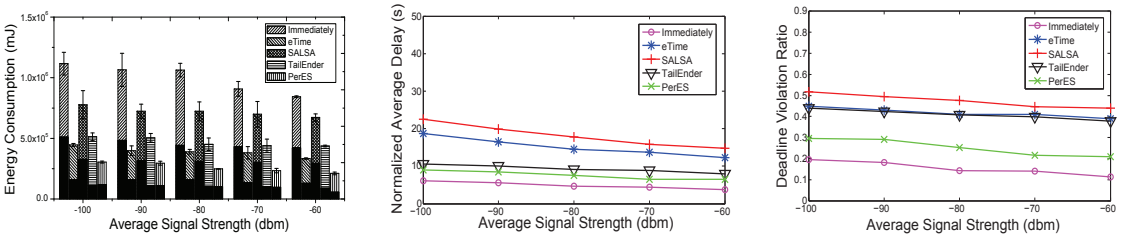
5.4.1 Impact of Arrival Pattern

In Fig. 12, as expected, heavier traffic results in larger energy cost (Fig. 12a) and performance cost (Fig. 12b and Fig. 12c). However, the distributions of the tail energy and data transmission energy have big difference among different schedulers. The tail energy of *SALSA* is on average twice that of *eTime*, *PerES* and *TailEnder*. This is because *SALSA* does not consider the tail energy in its decision. Further, because *TailEnder* is not aware of the signal variation, its



(a) Energy Consumption (b) Normalized Average Delay (c) Deadline Violation Ratio

Fig. 13: Performance over Different Task Deadline



(a) Energy Consumption (b) Normalized Average Delay (c) Deadline Violation Ratio

Fig. 14: Performance over Different Signal Strength

data transmission energy is on average twice that of other schedulers. *PerES*'s energy efficiency is similar to *eTime* and performs the best under different traffic conditions. Since both *PerES* and *TailEnder* are aware of the task deadline when scheduling, they have less normalized average delay than *eTime* and *SALSA*.

5.4.2 Impact of Task Deadline

In Fig. 13, we can see that *PerES* has the best energy efficiency. When increasing the minimum task deadline, the energy consumption of *TailEnder* increases significantly while *SALSA* gets an unstable energy output. Moreover, as Fig. 13b shows, when the deadline is small, both *TailEnder* and *PerES* achieve smaller average delay. However, when the deadline is large enough, they will generate larger average delay than *SALSA* and *eTime*. *SALSA* and *eTime* are not deadline-aware, while *TailEnder* and *PerES* intend to transfer tasks closer to their deadline to increase the energy efficiency at the cost of higher delay. The delay of *PerES* increases much slower than that of *TailEnder*. The deadline violation ratio of four schedulers decreases with increasing task deadline as expected. *PerES* achieves the lowest deadline violation ratio when the deadline is small and keeps a stable output when it turns large.

5.4.3 Impact of Signal Strength

In Fig. 14, the increase of average signal strength leads to performance improvement for all schedulers. Better signal leads to lower power and larger bandwidth for transmission and thus helps to reduce the energy and delay (Fig.14a and Fig.14b). *PerES* achieves the lowest energy and performance cost. Specially, for all schedulers, we can see that the transmission energy is reduced when the signal gets better while the tail energy does not have much change. A better signal requires a lower transmission power, which does not impact the tail power. However, it is also possible to obtain a lower tail energy when the signal turns better, e.g., -60dBm for *PerES*. This is because a better signal contributes to less transmission time and thus less possibility of violating the deadline (Fig.14c). Then it would provide more opportunity for a smart scheduler to aggregate the traffic.

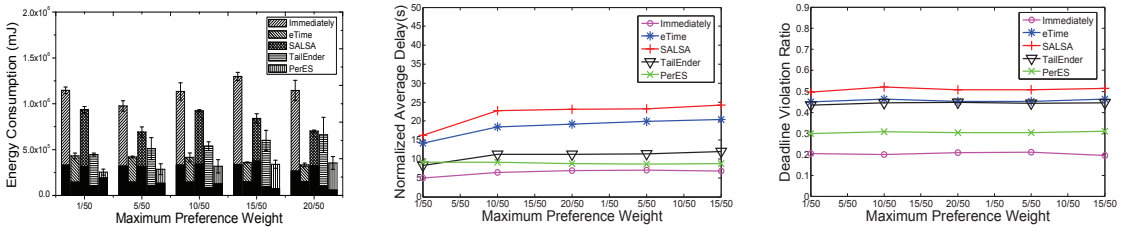
5.4.4 Impact of User Preference Weight

The impact of user preference is shown in Fig. 15. Since the user preference weight does not influence the scheduling procedure of *SALSA*, *eTime* and *TailEnder*, their energy consumption varies little while *PerES* achieves the best energy efficiency. The normalized average delay represents the delay from the user preference view. In Fig. 15b, the normalized average delay of *SALSA*, *eTime* and *TailEnder* increases with the maximum preference weight while *PerES* keeps a low and stable value. This is because they do not consider the user preference in different tasks. Since the average delay is normalized by the preference weights, with same task delay, the normalized average delay of these schemes will increase with the maximum preference weight. Further, *PerES* keeps the lowest deadline violation ratio.

5.5 Real-Traffic Application

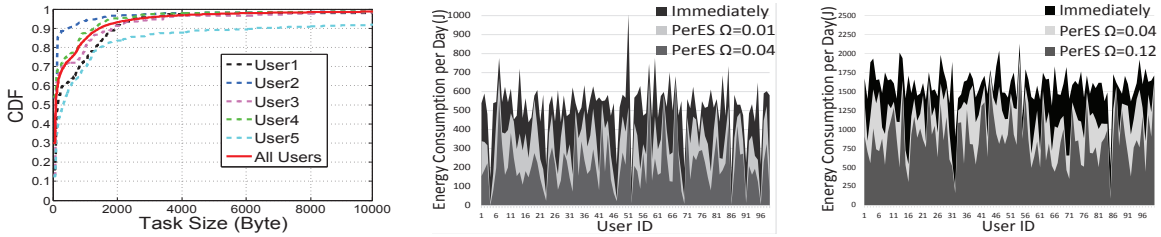
We analyzed a large traffic flow trace from 99 collection points by a 3G UMTS carrier in China on January 10, 2013. The trace data capture about 821 million flow records (1.2 Terabytes). Each record corresponds to the information of one flow which contains the user IP, server IP, flow time stamps, uploading and downloading data size but without any user data. Normally, the same user IP corresponds to one specific user within some time window. To simplify the process, we set the time window as one day. We consider the uploading data in one flow as one task generated from network applications on mobile devices and the same server IP as the server of one specific application.

As Fig. 16a shows, we first analyze the task size distribution in the traffic trace. We randomly choose one collection



(a) Energy Consumption (b) Normalized Average Delay (c) Deadline Violation Ratio

Fig. 15: Performance over Different User Preference Weight



(a) CDF of task size in real traffic scenarios (b) Energy for 100 users over 3G (c) Energy for 100 users over LTE

Fig. 16: Performance over Real Traffic Trace of 100 Users

point and pick the top 5 users that have the largest number of flows in one day for user-specific distribution. We further derive the flows of all users collected by all collection points in one day for a general distribution. We can find that, most users have the similar task size distribution, and small-size data account for a major portion of the trace, i.e., above 90% of tasks have their data size smaller than 6 Kbytes. This gives us the insight that most users have frequent task arrivals with a small data size, which will lead to a large fraction of tail energy.

As Fig. 16b shows, we randomly select 20 collection points and pick the top 5 users who own the largest number of flows in each collection point in one day. For each user, we select the top 5 applications that have the largest number of flows communicated with the user, and randomly select the signal trace in one day from the traces collected by real mobile users for 10 days. For the task flows of each user, we run both the default scheme *Immediately* and our *PerES* with different levels of performance requirements Ω on the phone for 100000 time slots. On average for each user, in the case $\Omega = 0.01$, *PerES* achieves totally 32% energy saving over the total transmission energy (tail energy reduction by 32.5%) with the normalized average delay as 20s and the deadline violation ratio as 0.22. In the other case $\Omega = 0.04$, *PerES* achieves totally 56% energy saving over the total transmission energy (tail energy reduction by 60.7%) with the normalized average delay as 57s and the deadline violation ratio as 0.31.

To further validate the performance of *PerES* over recent LTE networks, we apply the LTE energy model proposed in [6] to replace the previously utilized 3G energy model. The energy models in both networks are similar except the explicit parameter setting of the power and time of different radio states. To match the high throughput of LTE network, we equally increase the user traffic size by ten times and adjust the user performance bound Ω so that the user performance of LTE is comparable to that of 3G. As Fig. 16c shows, on average for each user, in the case $\Omega = 0.04$, *PerES* achieves 23.5% energy saving over the total transmission energy (reducing tail energy by 24.8%) with the normalized average delay to be 16s and the deadline violation ratio to be 0.08. In the other case $\Omega = 0.12$, *PerES* achieves 41.3% energy saving over the total transmission energy (with the tail energy reduced by 43.3%) with the normalized average delay to be 49s and the deadline violation ratio to be 0.13.

We can see that for comparable user performance, the energy saving percentage of LTE is reduced by about 10% compared to 3G. Further, we find that the percentage of the tail energy over the total LTE transmission energy is 10% larger than that of 3G. The reason is that the tail time of LTE (about 11 seconds) is longer than that of 3G and the tail power of LTE is also higher than 3G [3], [6]. Therefore, despite the presence of the DRX designed for energy saving, LTE is less energy efficient during the idle state and for transferring a small amount of data [3], which matches the feature of our trace in Fig. 16a.

6 Conclusion

In this paper, we propose adaptive scheduling algorithms to improve the energy efficiency of mobile devices in cellular networks while also considering user performance needs on multiple applications. Different from existing work, we formulate the hybrid energy optimization together with the performance degradation on user experience. We develop a practical online scheduler that can self-adapt to better meet user performance requirements and converge more than two times faster to the optimal energy consumption bound than that of state-of-art static schemes. Evaluation results demonstrate the effectiveness of our proposed schemes in achieving high energy approximation ratio to offline solutions and better performance compared to peer schemes. We further validate the energy efficiency of our proposed scheduling algorithm under different user experiences through a large amount of trace data collected.

ACKNOWLEDGMENTS

This work is supported by NSFC (no. 61422206, 61120106008), Tsinghua National Laboratory for Information Science and Technology (TNList). Xin Wang's research is supported by U.S. NSF CNS 1526843. This work is also partly supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117913].

REFERENCES

[1] 3GPP, "System impact of poor proprietary fast dormancy," *3GPP discussion and decision notes RP-090941*, 2009.
[2] P. Lee, T. Bu, and T. Woo, "On the detection of signaling dos attacks on 3g wireless networks," in *INFOCOM 2007*.
[3] J. Huang, F. Qian, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *MobiSys 2012*.
[4] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Top: Tail optimization protocol for cellular radio resource allocation," in *IEEE International Conference on Network Protocols (ICNP) 2010*.
[5] F. Yu, G. Xue, H. Zhu, Z. Hu, M. Li, and G. Zhang, "Cutting without pain: Mitigating 3g radio tail effect on smartphones," in *Mini Infocom 2013*.
[6] S. Deng and H. Balakrishnan, "Traffic-aware techniques to reduce 3g/lte wireless energy consumption," in *CoNEXT 2012*.
[7] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese, "Radiojockey: mining program execution to optimize cellular radio usage," in *MobiCom 2012*.
[8] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3g networks," in *IMC 2010*.
[9] J. Yeh, J. Chen, and C. Lee, "Comparative analysis of energy-saving techniques in 3gpp and 3gpp2 systems," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 1, pp. 432–448, 2009.
[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *IMC 2009*.
[11] H. Lagar-Cavilla, K. Joshi, A. Varshavsky, J. Bickford, and D. Parra, "Traffic backfilling: subsidizing lunch for delay-tolerant applications in umts networks," *ACM SIGOPS Operating Systems Review*, vol. 5, no. 3, pp. 77–81, 2012.
[12] H. Liu, Y. Zhang, and Y. Zhou, "Tailtheft: Leveraging the wasted time for saving energy in cellular communications," in *Proceedings of the sixth international workshop on MobiArch*. ACM, 2011, pp. 31–36.
[13] A. Pathak, Y. Hu, and M. Zhang, "Fine grained energy accounting on smartphones with eprof," in *EuroSys 2012*.
[14] A. Rahmati and L. Zhong, "Context-for-wireless: context-sensitive energy-efficient wireless data transfer," in *MobiSys 2007*.
[15] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang, "Experiences in a 3g network: interplay between the wireless channel and applications," in *MobiCom 2008*.
[16] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. Padmanabhan, "Bartendr: a practical approach to energy-aware cellular data scheduling," in *MobiCom 2010*.
[17] N. Ding, D. Wagner, X. Chen, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *SIGMETRICS '13*.
[18] B. Higgins, A. Reda, T. Alperovich, J. Flinn, T. Giuli, B. Noble, and D. Watson, "Intentional networking: opportunistic exploitation of mobile network diversity," in *MobiCom 2010*.
[19] M. Ra, J. Paek, A. Sharma, R. Govindan, M. Krieger, and M. Neely, "Energy-delay tradeoffs in smartphone applications," in *MobiSys 2010*.
[20] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "etime: energy-efficient transmission between cloud and mobile devices," in *Mini Infocom 2013*.
[21] 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*.
[22] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.
[23] S. Ren, Y. He, and F. Xu, "Provably-efficient job scheduling for energy and fairness in geographically distributed data centers," in *ICDCS 2012*.
[24] T. Zhang, X. Zhang, F. Liu, H. Leng, Q. Yu, and G. Liang, "etrain: Making wasted energy useful by utilizing heartbeats for mobile data transmissions," in *ICDCS 2015*.
[25] F. Liu, P. Shu, and J. Lui, "Appatp: An energy conserving adaptive mobile-cloud transmission protocol," 2015.
[26] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *Wireless Communications, IEEE*, vol. 20, no. 3, pp. 14–22, 2013.
[27] W. Hu and G. Cao, "Quality-aware traffic offloading in wireless networks," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*. ACM, 2014, pp. 277–286.
[28] F. Rebecchi, M. Dias de Amorim, V. Conan, A. Passarella, R. Bruno, and M. Conti, "Data offloading techniques in cellular networks: a survey," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 2, pp. 580–603, 2015.
[29] G. 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 *Wireless Conference 2009*.
[30] Y. Cui, S. Xiao, X. Wang, Z. Lai, Z. Yang, M. Li, and H. Wang, "Performance-aware energy optimization on mobile devices in cellular network," Tsinghua University, Tech. Rep., 2016. [Online]. Available: http://www.4over6.edu.cn/others/PerES_Tech.pdf
[31] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.
[32] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Periodic transfers in mobile applications: network-wide origin, impact, and optimization," in *WWW 2012*.

Yong Cui received the BE degree and the Ph.D. degree both on Computer Science and Engineering from Tsinghua University, China, respectively in 1999 and 2004. He is currently a full professor at the Computer Science Department in Tsinghua University. He published over 100 papers in the refereed conferences and journals with several Best Paper Awards. He co-authored 7 Internet standard documents (RFC) for his proposal on IPv6 technologies. His major research interests include mobile cloud computing and network architecture. He served or serves at the editorial boards on IEEE TPDS, IEEE TCC and IEEE Internet Computing. He is currently a working group co-chair in IETF.

Shihan Xiao received the B.Eng. degree in Electronic and Information Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012.

He is currently working towards his Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are in the areas of wireless networking and cloud computing.

Xin Wang received the B.S. and M.S. degrees in telecommunications engineering and wireless communications engineering respectively from Beijing University of Posts and Telecommunications, Beijing, China, and the Ph.D. degree in electrical and computer engineering from Columbia University, New York, NY. She is currently an Associate Professor in the Department of Electrical and Computer Engineering of the State University of New York at Stony Brook, Stony Brook, NY. Before joining Stony Brook, she was a Member of Technical Staff in the area of mobile and wireless networking at Bell Labs Research, Lucent Technologies, New Jersey, and an Assistant Professor in the Department of Computer Science and Engineering of the State University of New York at Buffalo, Buffalo, NY. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, as well as networked sensing and detection. She has served in executive committee and technical committee of numerous conferences and funding review panels, and serves as the associate editor of IEEE Transactions on Mobile Computing. Dr. Wang achieved the NSF career award in 2005, ONR challenge award in 2010.

Zeqi Lai is now a PhD student of the Department of Computer Science and Technology, Tsinghua University, China. His supervisor is Prof. Yong Cui. His research interests include cloud computing and cloud storage.

Zhenjie Yang received the BE degree in Networking Engineering from Dalian University of Technology, Liaoning, China, in 2015. He is currently a Ph.D. candidate in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include networking and cloud computing.

Minming Li received the B.Eng., M.Eng., and Ph.D. degrees from Tsinghua University, Beijing, China. He is currently an Associate Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include wireless networks, algorithms design and analysis, combinatorial optimization, scheduling, key management, and algorithmic game theory.

Hongyi Wang received the bachelors degree in computer science from Tsinghua University, China in 2009. He is working toward the master degree in the Department of Computer Science and Technology at Tsinghua University, supervised by Prof. Yong Cui. He received the Best Paper Award of ACM ICUIMC 2011. His research interests include data center networking, wireless networks, and mobile system.