Dynamic Speed Scaling for Energy Minimization in Delay-Tolerant Smartphone Applications

Jeongho Kwak†, Okyoung Choi†, Song Chong† and Prasant Mohapatra‡
†Department of Electrical Engineering, KAIST
‡Department of Computer Science, University of California, Davis
E-mail: {jh.kwak, okyoung}@netsys.kaist.ac.kr, songchong@kaist.edu, pmohapatra@ucdavis.edu

***Abstract*—Energy-delay tradeoffs in smartphone applications have been studied independently in dynamic voltage and frequency scaling (DVFS) problem and network interface selection problem. We optimize the two problems jointly to quantify how much energy can be saved further and propose a scheme called *SpeedControl* which jointly manages application scheduling, CPU speed control and wireless interface selection. The scheme is shown to be near-optimal in that it tends to minimize energy consumption for given delay constraints. This paper is the first to reveal energy-delay tradeoffs in a holistic view considering multiple wireless interfaces, DVFS and multitasking in smartphone. We perform real measurements on WiFi/3G coverage and throughput, power consumption of CPU and WiFi/3G interfaces, and CPU workloads. Trace-driven simulations based on the measurements demonstrate that SpeedControl can save over 30% of battery by trading 10 min delay as compared to existing schemes when WiFi temporal coverage is 65%, moreover, the saving tendency increases as WiFi coverage increases.**

I. Introduction

Energy consumption in processing and transferring data in smartphone is increasing. Maximum CPU clock frequency is consistently speeding up (e.g., the latest Qualcomm mobile chipset has maximum 2.5GHz CPU clock frequency [1]) to meet increasing demands of applications. Operation at maximum CPU clock frequency results in significant amount of energy consumption due to the fact that CPU power consumption is a super-linearly increasing function of clock frequency. Smartphones have multiple network interfaces including 3G and WiFi, which tend to facilitate more networking applications with higher data rates yielding higher networking energy consumption.

The most energy-consuming module in smartphones is display [2], [3]. According to recent power consumption survey of smartphones [2], LCD display of a smartphone consumes more than 800mW, which is significantly higher than what CPU or 3G/WiFi network interfaces consume. The power management of LCD display to save battery, however, is a technical challenge since reducing power consumption in LCD display directly affects quality of user experiences. The survey also reports that CPU and network interfaces consume most of the remaining power besides display when representative applications such as Angry birds or Web browser run. Notably, according to our measurements in section IV-B, the power consumption of CPU to process single bit for typical encoding applications is comparable with that of WiFi or 3G interface to transmit single bit. Thus, power management of CPU and network interfaces are equally important for a given fixed LCD display power management, and must be optimized jointly if they are coupled to each other.

Many application processors (APs) for smartphones support Dynamic Voltage and Frequency Scaling (DVFS) [1], [4], [5], which controls CPU clock frequency and voltage depending on CPU workloads. DVFS exploits power saving opportunities owing to the fact that CPU power consumption depends super-linearly on CPU clock frequency, but power saving comes at the cost of increasing delay.

For energy minimization in a smartphone with multiple network interfaces, previous works [6], [7] proposed energy-efficient network selection policies under heterogeneous wireless network interfaces (e.g., 3G and WiFi interfaces) for delay-tolerant applications. They studied tradeoffs between energy saving and delay in data transmission by intentionally deferring data transmission until the device meets an energy-efficient network. To the best of our knowledge, however, no works have studied joint optimization of CPU speed scaling (i.e., DVFS policy) and network speed scaling (i.e., network selection policy) for energy minimization.

Independent control of CPU speed and network selection causes energy inefficiency. For instance, consider a situation where a smartphone runs an application which processes a file (e.g., encoding or transcoding a video file) and then uploads the encoded file to a cloud server. Assume that the network condition is bad, say, only 3G link with low data rate is available. As CPU workload increases in this situation, DVFS would keep increasing the data processing speed accordingly so that the network queue would become a bottleneck with large backlog eventually since the data processing speed would exceed the low data transmission speed at some point. This is exactly the situation that one would like to avoid from energy-efficiency point of view because CPU would operate at an unnecessarily fast speed wasting energy in this situation. On the other hand, if the backlog at the network queue were small, one could postpone data transmission until the device finds out an energy-efficient network, say, WiFi. Note that average energy consumption in transmitting a single bit in WiFi links is much less than that in 3G links (see our measurements in

This research was funded in part by the MSIP(Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013 and the NSF Grant CNS-1319721 and the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE
978-14799-3360-0/14/$31.00 ©2014 IEEE

section IV-B). Therefore, the situation we consider above is also harmful in terms of energy efficiency because the network queue should transmit data over the energy-inefficient link due to the large backlog and cannot wait for an energy-efficient network to come.

According to recent survey on top 50 smartphone applications in Google Play [8], 44% are networking applications (NAs) and 56% are non-networking applications (NNAs), and contemporary smartphone operating systems provide multitasking capability (e.g., iOS and Android). Therefore, many smartphones are likely to run both NAs and NNAs simultaneously. NAs use both CPU and network resources whereas NNAs use CPU resource only. If NAs and NNAs share the CPU queue, they would interfere with each other. A way to isolate the performance of NNAs from that of NAs should be considered in the joint optimization of CPU and network speed scaling.

In this paper, we propose a dynamic speed scaling scheme called SpeedControl which jointly adjusts both processing speed and networking speed by three control knobs: application scheduling, CPU speed control and network (wireless interface) selection. By invoking the Lyapunov drift plus penalty method [9], the scheme is shown to be near-optimal in the sense that it minimizes total energy consumption of CPU and network interface for given delay constraints. This paper is the first to reveal energy-delay tradeoffs in energy minimization in a holistic view for smartphones with multiple wireless interfaces, DVFS and multitasking capability. In order to create realistic simulation scenarios, we perform real measurements or use public measurement traces on WiFi/3G coverage and data rate on the metropolitan area in South Korea, power consumption of CPU and WiFi/3G interfaces in popular smartphone models, and file size distribution and CPU workloads for popular video clips.

The contribution of the paper is summarized as follows.

- The proposed SpeedControl scheme is the first to jointly optimize CPU speed and network speed scaling for energy minimization in delay-tolerant smartphone applications, and is shown to be near-optimal in that it tends to minimize energy consumption for given delay constraints.
- The SpeedControl is the first to address the co-existence issue of NAs and NNAs sharing CPU resource in smartphones with multitasking capability. In order to isolate the performance of NNAs from that of NAs subject to the SpeedControl in this highly coupled environment, an application scheduling policy to determine the sequence of CPU access between NAs and NNAs is incorporated into the SpeedControl scheme.
- The SpeedControl not only significantly outperforms existing schemes but also do not affect the performance of background NNAs. Trace-driven simulations based on real measurements of operational environments and system parameters demonstrate that the SpeedControl scheme can save over 30% of smartphone battery by trading about 10 min transfer delay as compared to existing schemes when WiFi temporal coverage is about 65%. Moreover, the saving tendency increases as WiFi temporal coverage increases.

In the rest of this paper, we begin with related work in Section II. In Section III, we describe the system model and problem formulation. And then, we propose SpeedControl algorithm. Then, in Section IV, we explain the design of the SpeedControl algorithm. Next, in Section V, we extensively evaluate our SpeedControl algorithm by theoretical analysis, measurement and trace driven simulation and experiment. Finally, we conclude this paper in Section VI.

A. Related Work

There have been extensive studies on energy-delay tradeoff in network devices such as cellular base station or router using DVFS [10]–[12]. Son *et. al.* [10] suggest energy efficient joint control of DVFS and user association using the fact that power consumption of base station is well modeled by a cubic polynomial scaling of processing speed. Andrews *et. al.* [12] study a routing problem with the objective of provisioning guaranteed speed/bandwidth for a given processing demand while minimizing energy consumption of network-wide routing devices.

Several DVFS techniques [1], [4], [5] have been considered in mobile devices. Chen *et. al.* [4] suggested energy-efficient task scheduling scheme for the real time workloads in multiprocessor dynamic voltage scaling (DVS) systems. Also, Liang *et. al.* [5] showed that there exists a critical CPU clock speed to minimize the energy consumption of handheld devices. Recent mobile chipsets, e.g., Snapdragon S4 [1], have also adopted a DVFS technique for the energy efficiency of devices. However, their DVFS policies are far from an optimal one and inefficient, e.g., in Ondemand policy [13], CPU speed is set to be maximum when workload is over a certain threshold, which is controlled manually, and then gradually decrease the speed depending on the workload. To the best of our knowledge, there have been no effort factoring in the wireless network dynamics and heterogeneity of applications in the context of mobile device DVFS.

Recently, the interests on network selection considering battery consumption of smartphone, which include [6], [7], [14], [15], have been increasing. Rahmati *et. al.* [14] suggest on-the-spot network selection by examining tradeoff between energy consumption for WiFi search and transmission efficiency when WiFi network is intermittently available. Some studies [6], [7] suggest delayed network selection by exploring tradeoff between transmit power of heterogeneous network interfaces (3G, WiFi) and transmission delay. Lee *et. al.* [15] show 20% power saving can be achieved by permitting 1 hour delayed WiFi offloading for application having strict delay constraint.

II. Energy-efficient SpeedControl Algorithm

In this section, we formulate an optimization problem considering energy minimization with queueing stability, and propose an energy-minimal joint application scheduling, CPU speed adjustment, and network interface selection, called SpeedControl algorithm.

A. Model and Problem Formulation

Arrival Model. We consider two types of delay-tolerant applications. One is networking application which uses both processing and networking resources and the other is non-networking application which uses only processing resource in a smartphone. For instance, a smartphone user records two video clips using camera application. The one of recorded video clip is encoded and uploaded to cloud server such as Dropbox (networking application, NA), and the other video clip is just encoded in the smartphone (non-networking application, NNA). Let a set of applications be $\mathcal{K} = \{1, ..., K\}$. For simplicity, we consider one NA and another NNA. However, it can be easily generalized to more applications. We consider a time-slotted system indexed by $t = \{0, 1, ...\}$ where the interval is Δt. For each time slot t, workload $A_{NA}(t)$ for NA, and $A_{NNA}(t)$ for NNA are arrived, respectively.

CPU & Network Model. We assume that a smartphone has one CPU core which handles several applications running in the smartphone. The workload of each application demands different CPU processing resource. We call this notion by *processing density* (in cycles/bit) γ_{NA} for NA, γ_{NNA} for NNA which are defined as the average number of CPU cycles required per bit when the application is processed by CPU [16]. We assume that the smartphone can adjust CPU speed $s(t) \in \{s_1, s_2, ...s_{max}\}$ (in cycles/Δt) every time slot t. The levels of CPU speed are determined by Kernel in operating system of the smartphone. For network side, achievable uplink throughput $\mu_l(t)$ for different network l varies along with dynamics of wireless channel states every time slot t. Also, we assume that the smartphone can select among no network transmission (N), cellular (C) and WiFi (W) networks every time slot t, i.e., $l(t) \in \{N, C, W\}$.

Queueing Model. We consider a queueing model as illustrated in Fig. 1. The queueing model is designed to prevent a situation where NNA has a penalty when network is the bottleneck. NNA should not be affected by the network environment since it does not use network resource. However, if we design queueing model that NA and NNA packets are not segregated for processing, the system cannot address this criterion when network is the bottleneck and NNA workload is located behind than NA workload in CPU queue. In this case, NNA workload cannot be scheduled even though NNA does not use network resource. This is the reason why NA and NNA workloads should be distinguished and processed separately in our queueing model. For the queueing model, $Q^c(t)$ denotes total CPU queue length at time slot t, $Q^c_{NA}(t)$ and $Q^c_{NNA}(t)$ denote CPU queue lengths for NA and NNA at time slot t, respectively, i.e., $Q^c(t) = Q^c_{NA}(t) + Q^c_{NNA}(t)$. Also, $Q^n(t)$ denotes network queue length at time slot t. For application scheduling, $\theta(t) \in \{0, 1\}$ denotes scheduling indicator by CPU processor at time slot t, e.g., if $\theta(t)$ is 0, NNA is scheduled. We assume that the unit of queue lengths is a bit, thus the CPU speed $s(t)$ (in cycles/Δt) should be divided by processing density (cycles/bit) γ_{NA} or γ_{NNA} for unit agreement. Then, queue lengths of all CPU and network queues are updated as follows.

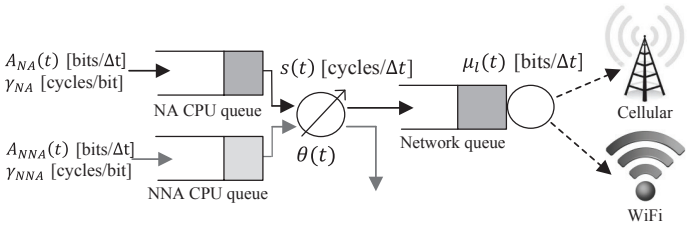


Fig. 1: Queueing model for SpeedControl system

$$Q^c_{NA}(t+1) = \left[Q^c_{NA}(t) - \frac{\theta(t)s(t)}{\gamma_{NA}} + A_{NA}(t)\right]^+ \text{[bits]}$$
$$Q^c_{NNA}(t+1) = \left[Q^c_{NNA}(t) - \frac{(1-\theta(t))s(t)}{\gamma_{NNA}} + A_{NNA}(t)\right]^+ \text{[bits]} \quad (1)$$
$$Q^n(t+1) = \left[Q^n(t) - \mu_l(t) + \theta(t) \min\left\{Q^c_{NA}(t), \frac{s(t)}{\gamma_{NA}}\right\}\right]^+ \text{[bits]}$$

Power Model. CPU power consumption can be modeled in general as follows [12].

$$P^c(s(t)) = \alpha s(t)^3 + \beta \quad (2)$$

where α and β are constants depending on different smartphones. Since the network power depends on the selected network [6], the network power is modeled as follows.

$$P^n(l(t)) \in \{P^n(N), P^n(C), P^n(W)\} \quad (3)$$

These power consumption models will be further addressed by real measurement in Section IV.

Problem Formulation. Our objective under the queueing model in Fig. 1 is to develop energy-minimal joint application scheduling, CPU speed adjustment and network interface selection policies. The smartphone can serve all arrival workloads of NA and NNA within the capacity region which is the set of all acceptable arrival rates with guaranteeing stability of CPU and network queues. The optimization problem is chosen such that

$$(\mathbf{P}): \quad \min \bar{P} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} (P^c(t) + P^n(t)), \quad (4)$$

$$s.t. \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{Q^c(\tau) + Q^n(\tau)\} < \infty, \quad (5)$$

where $Q^c(t) = Q^c_{NA}(t) + Q^c_{NNA}(t)$. The constraint means that average CPU and network queue lengths should be finitely maintained [9]. For the above problem **(P)**, we determine the application scheduling $\theta(t)$, CPU speed $s(t)$ and network interface selection $l(t)$ every time slot t.

B. Application Scheduling, CPU Speed and Network Interface Selection (SpeedControl) Algorithm

We obtain the solution of our problem **(P)** under unknown future sequences of wireless network states and data arrival by invoking Lyapunov drift plus penalty technique [9]. Our SpeedControl algorithm is to jointly control application scheduling, CPU speed adjustment and network selection so as to satisfy the decomposed objectives for each control variable.

Algorithm Description. The SpeedControl algorithm is described as follows.

SpeedControl Algorithm

For every time slot t,
if $Q^n(t) \geq \mu_{max}(t)$, **then**
  Schedule NNA ($\theta^*(t) = 0$)
  Select CPU speed $s^*(t)$ by

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - \frac{s(t)}{\gamma_{NNA}} Q^c_{NNA}(t) \right\} \quad (6)$$

  Select network $l^*(t)$ by

$$\min_{l^*(t)} \left\{ VP^n(l(t)) - \mu_l(t)(Q^c_{NA}(t) + Q^n(t)) \right\} \quad (7)$$

else
  if $\frac{Q^c_{NA}(t)+Q^n(t)}{\gamma_{NA}} \geq \frac{Q^c_{NNA}(t)}{\gamma_{NNA}}$, **then**
    Schedule NA ($\theta^*(t) = 1$)
    Select CPU speed $s^*(t)$ by

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - \frac{s(t)}{\gamma_{NA}} (Q^c_{NA}(t) + Q^n(t)) \right\} \quad (8)$$

    Select network $l^*(t)$ by (7)
  elseif $\frac{Q^c_{NA}(t)+Q^n(t)}{\gamma_{NA}} < \frac{Q^c_{NNA}(t)}{\gamma_{NNA}}$, **then**
    Schedule NNA ($\theta^*(t) = 0$)
    Select CPU speed $s^*(t)$ by (6)
    Do not select network ($l(t) = N$)
end all

where $\mu_{max}(t)$ is the maximum uplink throughput among available networks at time slot t.

The problems to select CPU speed and network (6)-(8) can be interpreted as follows. An energy-delay tradeoff can be controlled by single parameter V, i.e., as V increases, energy can be saved by trading longer delay. Also, the first term with V of problems (6)-(8) strives to minimize CPU or network power consumption, and remained terms without V strives to stabilize CPU and/or network queue. If V is small and queues increase due to the workload arrival, the system strives to immediately reduce queue lengths by increasing CPU speed $s(t)$ or selecting immediately available network, i.e., the system is sensitive to queue variation. However, the CPU or network power consumptions increase due to the high variation of $s(t)$ and the fact that the system is likely to immediately transmit with a little interest on energy-efficiency of network interface (e.g., 3G with low throughput). Therefore, the system makes more effort to reduce queue length than to reduce energy. On the other hand, if V is large and queues increase due to the workload arrival, the system does not sensitively react to increment of queues due to higher weight on the first term with V than remained terms without V. However, for the same workload, the system controls $s(t)$ more smoothly and is likely to defer transmission or select energy-efficient networks (e.g., WiFi network). Therefore, the problems (6)-(8) make more effort to reduce CPU and network energy than to reduce queue length.

Scheduling of NA and NNA depends on the following conditions. *(i)* If $Q^n(t) \geq \mu_{max}(t)$, network is the bottleneck. In this case, CPU part strives to reduce the queue length of NNA than NA because NNA should not be affected by network environment, hence always NNA is scheduled. On the other hand, *(ii)* if $Q^n(t) < \mu_{max}(t)$, network is not bottleneck. In this case, CPU part strives to reduce the queue length of NA and NNA as fairly as possible since NNA does not be affected by network environment, so the system schedules application by comparing queue lengths and processing densities of NA and NNA.

III. Algorithm Design

We derive the SpeedControl algorithm in inheriting Lyapunov drift plus penalty function [9].

Making Single Objective. Our original objective **(P)** is to minimize CPU and network power consumption with satisfying queueing stability. First, we define Lyapunov function and Lyapunov drift function as follows.

$$L(t) \triangleq \tfrac{1}{2}[Q^c_{NA}(t) + Q^n(t)]^2 + \tfrac{1}{2}[Q^c_{NNA}(t)]^2 \quad (9)$$

$$\Delta(L(t)) \triangleq \mathbb{E}\{L(t+1) - L(t) | \mathbf{Q}(t)\} \quad (10)$$

$$\mathbf{Q}(t) = \{Q^c_{NNA}(t), Q^c_{NA}(t), Q^n(t)\} \quad (11)$$

The Lyapunov function (9) is designed to fairly stabilize NA queues ($Q^c_{NA}(t) + Q^n(t)$) and NNA queue ($Q^c_{NNA}(t)$). This is a key design principle to isolate NNA performance from that of NA in terms of delay. Under the Lyapunov function design, if the network queue is accumulated due to the network bottleneck, scheduling NNA would not reduce NA queues ($Q^c_{NA}(t) + Q^n(t)$), so NNA would be always scheduled to reduce NNA queue ($Q^c_{NNA}(t)$). On the other hand, if the network queue is not accumulated, NA and NNA would be fairly scheduled in terms of queue lengths[1].

Next, we define drift plus penalty function where the penalty function is the sum of expected CPU and network power consumption during time slot t ($\mathbb{E}\{P^c(s(t))|\mathbf{Q}(t)\} + \mathbb{E}\{P^n(l(t))|\mathbf{Q}(t)\}$) in inheriting drift-plus-penalty expression [9] as follows.

$$\Delta(L(t)) + V\mathbb{E}\left\{ P^c(s(t)) + P^n(l(t)) | \mathbf{Q}(t) \right\} \quad (12)$$

where V is energy-delay tradeoff parameter. Then, our single objective is to minimize the equation (12).

Deriving Upper Bound. Next, we assume that workload arrival $A_{NA}(t)$ and $A_{NNA}(t)$, CPU speed $s(t)$ and uplink throughput $\mu_l(t)$ for all available networks are bounded as follows.

$$A_{NA}(t) \leq A_{NA,max}, \quad A_{NNA}(t) \leq A_{NNA,max} \quad (13)$$

$$s(t) \leq s_{max}, \quad \mu_l(t) \leq \mu_{max} \quad (14)$$

These bounded values and queueing dynamic equations in (1) make drift plus penalty function (12) be bounded as a following Lemma.

[1]In fact, since the units of processing speed (in cycles/Δt) and traffic in the queue (in bits) are different, processing density (in cycles/bit) of application should be also considered.

Lemma 1. *Under any possible control variables $\theta(t) \in \{0,1\}$, $s(t) \in \{s_1, s_2, ..., s_{max}\}$ and $l(t) \in \{N, C, W\}$, we have:*

$$\Delta(L(t)) + V\mathbb{E}\Big\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\Big\} \leq$$
$$B + V\mathbb{E}\Big\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\Big\}$$
$$- \mathbb{E}\Big\{Q^c_{NNA}(t)\Big(\frac{(1-\theta(t))s(t)}{\gamma_{NNA}} - A_{NNA}(t)\Big)|\mathbf{Q}(t)\Big\} \quad (15)$$
$$- \mathbb{E}\Big\{(Q^c_{NA}(t) + Q^n(t))\times$$
$$\Big(\min\Big\{\frac{\theta(t)s(t)}{\gamma_{NA}} + Q^n(t), \mu_l(t)\Big\} - A_{NA}(t)\Big)|\mathbf{Q}(t)\Big\},$$

where $B = \frac{1}{2}(\frac{s^2_{max}}{\gamma_{NNA}} + \mu^2_{max} + A^2_{NA,max} + A^2_{NNA,max})$.

Proof: Please refer to our technical report [17]. ∎

Deriving Solution. Minimizing the left-hand side of (15) means that our original problem **(P)** is satisfied. We show that the problem **(P)** has an optimal policy π^* at the following Theorem 1.

Theorem 1. *For any mean arrival workload $\mathbb{E}\{A_{NA}(t)\} = \lambda_{NA}, \mathbb{E}\{A_{NNA}(t)\} = \lambda_{NNA}$ within capacity region, $\lambda_{NA} + \lambda_{NNA} \in \Lambda$, where Λ denotes all mean arrival workloads that the smartphone can process within finite time, there exists a stationary randomized control policy π^* that selects application scheduling $\theta(t)$, CPU speed $s(t)$ and network $l(t)$ every time slot t satisfying the following equations:*

$$\mathbb{E}\{P^c(s(t)^{\pi^*})\} = \overline{P^c}(\lambda_{NA} + \lambda_{NNA}) \quad (16)$$
$$\mathbb{E}\{P^n(l(t)^{\pi^*})\} = \overline{P^n}(\lambda_{NA}) \quad (17)$$
$$\mathbb{E}\Big\{\frac{\theta(t)^{\pi^*}s(t)^{\pi^*}}{\gamma_{NA}}\Big\} = \mathbb{E}\{\mu_l(t)^{\pi^*}\} \quad (18)$$
$$\mathbb{E}\{A_{NA}(t)\} = \mathbb{E}\Big\{\frac{\theta(t)^{\pi^*}s(t)^{\pi^*}}{\gamma_{NA}}\Big\} \quad (19)$$
$$\mathbb{E}\{A_{NNA}(t)\} = \mathbb{E}\Big\{\frac{(1-\theta(t)^{\pi^*})s(t)^{\pi^*}}{\gamma_{NNA}}\Big\} \quad (20)$$

where $\overline{P^c}(\lambda_{NA} + \lambda_{NNA})$ and $\overline{P^n}(\lambda_{NA})$ are average CPU and network power consumption to process $\lambda_{NA} + \lambda_{NNA}$ and transmit λ_{NA}, respectively.

Proof: It can be proven using Caratheodory's theorem in [18]. ∎

Then, an optimal algorithm (OPT) which minimizes (12) is to find control variables ($\theta(t), s(t), l(t)$) which minimize the left-hand side of (15), i.e., the optimal algorithm makes the right-hand side of (15) the smallest one among the values which obtains from all possible stationary randomized control policies. Then, the right-hand side of (15) can be decomposed into three problems to find application scheduling $\theta(t)$, CPU speed $s(t)$ and network selection $l(t)$ when network is the bottleneck ($Q^n(t) \geq \mu_{max}(t)$). However, since finding the optimal control variables is tightly coupled with all control variables when the network is not bottleneck ($Q^n(t) < \mu_{max}(t)$), we design simple and decoupled SpeedControl algorithm by taking some approximation and assumptions. First of all, we take following reasonable approximation.

$$Q^n(t) \approx 0, \text{ for } Q^n(t) < \mu_{max}(t) \quad (21)$$

Then, if $Q^c_{NNA}(t)\frac{s(t)}{\gamma_{NNA}} > Q^c_{NA}(t)\frac{s(t)}{\gamma_{NA}}$, the right-hand side of (15) can be decomposed into ($\theta(t) = 0, s(t), l(t) = N$) such as (6). In the other case, however, we should address complicated problems as follows.

$$\min_{s(t)} V\big(P^c(s(t)) + P^n(N)\big) - \frac{s(t)}{\gamma_{NNA}}Q^c_{NNA}(t) \quad (22)$$
$$\min_{s(t),l(t)} V\big(P^c(s(t)) + P^n(l(t))\big) - \min\Big(\frac{s(t)}{\gamma_{NA}}, \mu_l(t)\Big)Q^c_{NA}(t) \quad (23)$$

If (22) $\geq$ (23), then $\theta(t) = 1$ and if (22) $<$ (23), then $\theta(t)$=0. Since jointly solving (22) and (23) is complicated and hard to solve, we assume that always NA is scheduled ($\theta(t)$=1), and assume $\frac{\theta(t)s(t)}{\gamma_{NA}} < \mu_l(t)$ for CPU speed selection, $\frac{\theta(t)s(t)}{\gamma_{NA}} \geq \mu_l(t)$ for network selection in order to prevent the situation that the network queue is empty. Then, SpeedControl algorithm selects CPU speed and network interface depending on the network bottleneck and application scheduling. Although our SpeedControl algorithm takes some approximation and assumptions, the algorithm shows the nearly the same performance with the optimal algorithm in further simulation results.

IV. Performance Evaluation

In this section, we evaluate the performance of proposed algorithm through theoretical analysis, measurements and trace-driven simulations and experiments.

A. Theoretical Analysis

We describe the performance of an optimal algorithm by theoretical analysis. The sum of NA and NNA queue lengths and the sum of average CPU and network power consumption can be upper bounded by following Theorem 2, respectively.

Theorem 2. *Let $t = \{0, 1, ...T-1\}$. Suppose there exist $\epsilon' > 0$ and $\epsilon'' > 0$ such that $\lambda_{NA} + 2\epsilon' \in \Lambda_{NA}$ and $\lambda_{NNA} + \epsilon'' \in \Lambda_{NNA}$, then under the optimal algorithm (OPT), we have:*

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\{Q^c(t) + Q^n(t)\} \leq \frac{B + VP_{max}}{\epsilon}, \quad (24)$$

$$\bar{P}^{c,OPT} + \bar{P}^{n,OPT} \leq P^*(2\epsilon) + \frac{B}{V}. \quad (25)$$

where $Q^c(t) = Q^c_{NA}(t) + Q^c_{NNA}(t)$, $\epsilon = 2\epsilon' = \epsilon''$, $P_{max} = P^c_{max} + P^n_{max}$ is the maximum average CPU and network power consumption and $P^*(2\epsilon) = P^{c^*}(2\epsilon) + P^{n^*}(\epsilon)$ is the optimal lower bound of CPU and network power consumption.

Proof: Please refer to our technical report [17]. ∎

The result of Theorem 2 can be interpreted as follows. As the energy-delay tradeoff parameter V decreases, the sum of average queue lengths (NA and NNA) decreases whereas the average power consumption increases. On the other hand, as V is larger, the average power consumption decreases whereas the sum of average queue lengths increases. When V goes to

infinity, the smartphone consumes the optimal average power $P^*(2\epsilon)$.

B. Measurement and Trace-Driven Simulation

In this section, we verify the SpeedControl algorithm by real power measurements and trace-driven simulations under several environments.

Real Power Measurement. We measure and analyze the CPU power consumption over different CPU speed for five android smartphones. We connect Monsoon power monitor [19] to experimental smartphones and measure the real power consumption of the five smartphones. Because the power consumption of CPU module cannot be directly measured, we turn off the other controllable modules. Also, for 100% utilization of CPU, we run a video encoding application[2]. Five smartphones have different Kernel and OS, so the CPU clock frequency-voltage matching tables of five smartphones are different, respectively. For example, Nexus S has Trinity Kernel and Android 4.0 OS, so it has six levels of CPU speeds and each CPU speed matches with specific voltage (100MHz-975mV, 200MHz-975mV, 400MHz-1025mV, 800MHz-1250mV, 1000MHz-1450mV, 1440MHz-1500mV). Fig. 2 depicts the measured CPU power consumption as a function of CPU speed for five different smartphones. The diagrams (e.g., circles, square, triangle, star) are the real power measurement values for different discrete CPU speed levels. The measured discrete power consumptions are well modeled by a cubic polynomial scaling of speed to power for all smartphones. From these results, we can make the CPU power consumption model as a function of CPU speed as follows.

$$P^c(s(t)) = \alpha s(t)^3 + \beta \tag{26}$$

where $s(t)$ is CPU speed, t is time slot of which interval is minimum static CPU speed period, α and β are constant values depending on the different smartphones.

Also, we measure the power consumption of 3G and WiFi network interfaces using Monsoon power monitor [19] for two different android smartphones, respectively. For network interface measurement, we turn off running applications in a smartphone except for throughput measurement application which is made by us for transmitting dummy data to the server. We make an application to measure the throughput in server, and then, measure the power consumption while the application transmits 2MByte (WiFi) and 50KByte (3G) dummy data to the server. The measured network interface power consumption of two smartphones can be shown in Table II, which provides following findings. *(i)* the 3G and WiFi transmit powers are the similar in both smartphones, yet *(ii)* WiFi transmit power in W/bit is much smaller than that of 3G. This implies that WiFi network is more energy efficient than 3G network for transmitting same quantity of data. Also, *(iii)* average transmit power consumptions of WiFi and 3G interfaces in W/bit are comparable with

[2]In our measurement, all smartphones has 100% utilization for all CPU speed levels when running video encoding application. Also, since we run only one application, only one CPU core can be operated in spite of dual core CPU.

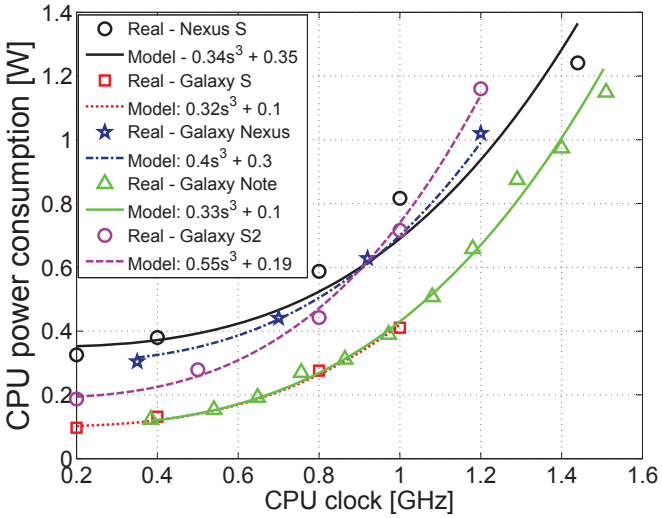


Fig. 2: CPU power consumption vs. CPU speed

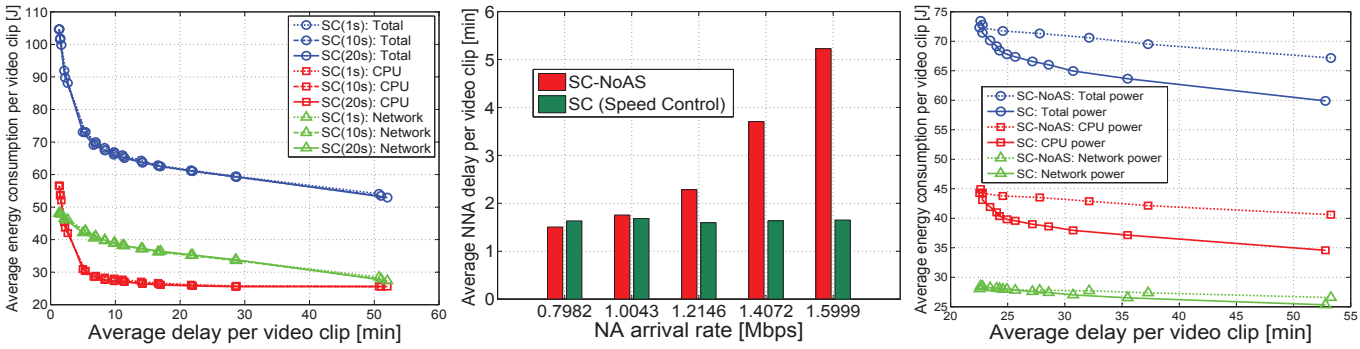
Nexus S				
	idle(mW)	transmit(mW)	uplink throughput(Mbps)	average tx. power in W/bit
WiFi	230	702±72.5	1.66-3.12	308.6×10^{-9}
3G	213	1217±185	0.69-0.85	1700×10^{-9}
Galaxy Nexus				
	idle(mW)	transmit(mW)	uplink throughput(Mbps)	average tx. power in W/bit
WiFi	99	875±22	6.36-6.87	134×10^{-9}
3G	82	964±210	0.354-0.742	1951×10^{-9}

TABLE I: Network power consumptions for two smartphones

CPU power consumption in W/bit (CPU power consumption in W/bit assuming that CPU speed is 1GHz and processing density is 1000cycles/bit: 816.5×10^{-9}W/bit (Nexus S), 688.1×10^{-9}W/bit (Galaxy Nexus)). It supports the fact that power management of CPU and network interfaces are equally important for processing or transmitting the same quantitiy of data.

Real Traces. We collect workload arrivals, processing densities of NA and NNA and uplink throughputs of 3G and WiFi networks. First, we use open dataset of real YouTube video data size from [20] to generate workload arrival traces of NA and NNA. Second, we run an encoding application in a smartphone for several video clips and measure their completion times, respectively. Then, we compute processing densities by dividing processing quantity (in cycles) with the size of a video clip (in bits). The measured processing densities are between 200cycles/bit to 1200cycles/bit for different video formats and clips. Third, we measure the 3G and WiFi uplink throughputs and WiFi connectivity of 5 smartphones for 2 weeks in a metropolitan area of South Korea. Using our uplink measurement application, a smartphone transmits dummy data to 3G and WiFi networks for every 20 seconds, respectively, and records WiFi connectivity logs. Then, server application measures the uplink throughput of all smartphones. Measured average uplink throughput for 3G and WiFi are 0.76Mbps and 3.01Mbps, respectively, and the average WiFi temporal coverage is 63% in daytime (9:00AM to 9:00PM).

Setup. We consider a scenario that two applications are running in a smartphone: *(i)* In networking application (NA), a video



(a) Impact of time scale difference: WiFi temporal coverage - 50%, γ_{NA}=300cycles/bit, γ_{NNA}=1200cycles/bit

(b) Average delay of NNA in network bottleneck case: WiFi temporal coverage - 0%, average network throughput=0.76Mbps, γ_{NA}=600cycles/bit, γ_{NNA}=600cycles/bit

(c) Energy-delay tradeoff in network bottleneck case: WiFi temporal coverage - 0%, average network throughput=0.76Mbps, γ_{NA}=600cycles/bit, γ_{NNA}=600cycles/bit

Fig. 3: Characteristic evaluation of SpeedControl

clip generated in the smartphone is encoded and transmitted to the cloud server. *(ii)* In non-networking application (NNA), another video clip generated in the smartphone is just encoded. During 1 second, the $A_{NA}(t)$ and $A_{NNA}(t)$ size video clips are independently generated with 0.8% probability. We use the one of CPU power-speed sets in Fig. 3, and use transmit and idle powers for 3G and WiFi interfaces in Table II. Control intervals are 1 second for application scheduling and CPU speed adjustment, and 20 seconds for network selection. This is reasonable setting since the associated network cannot be changed as fast as CPU speed adjustment due to the vertical handover delay. For uplink throughput estimation in simulation[3], we suppose the current uplink throughput as the uplink throughput obtained in just before time slot. If the device does not transmit data through the corresponding network at just before time slot, we use time average throughput of the corresponding network. Performance metrics are average CPU and/or network energy consumption per video clip and the average delay of NA and NNA per video clip. We compare DVFS+SALSA and Max+SALSA with our SpeedControl algorithm. DVFS+SALSA is conventional DVFS[4] with delayed network selection [6], and Max+SALSA uses always maximum CPU speed with delayed network selection.

Observations. From the simulation results, we obtain the interesting observations as follows.

Impact of time scale difference. We verify the impact of the time scale difference between CPU speed adjustment and network selection. Fig. 3(a) depicts the energy-delay tradeoff of SpeedControl for several time scales of network selection. This figure shows that the average energy consumption and delay performance of SpeedControl for different network selection time scales (1 second, 10 seconds, 20 seconds) are almost the same. This is because that the time scale of WiFi availability in our trace is much larger than that of network selection (20 seconds). Therefore, we henceforth use 20 seconds for the time interval of network selection in the remained simulations.

Impact of application scheduling. To verify that SpeedControl well isolates the performance of NNA from that of NA in terms of delay, we carry out the simulation in the network bottleneck case. For comparison, we consider an algorithm without application scheduling control, called SC-NoAS. This algorithm schedules the application by first in first out (FIFO) manner, but CPU speed adjustment and network selection are operated like SpeedControl. Fig. 3(b), 3(c) depict delay performance of NNA and energy-delay tradeoff of SC-NoAS and SpeedControl when network is the bottleneck. In simulation for Fig. 3(b), the arrival rate of NNA is the same (0.8Mbps), but the arrival rate of NA is increased up to twofold. Fig. 3(b) shows that SpeedControl guarantees average delay of NNA when the arrival rate of NA is increased, whereas SC-NoAS increases average delay of NNA even though the arrival rate of NNA is not increased. This implies that SpeedControl makes NNA not be influenced by network environment, which is also related with design issue of SpeedControl in Section III. As a result, SpeedControl has better delay performance and energy consumption than SC-NoAS as shown in Fig. 3(c).

Energy-delay tradeoff. Fig. 4 depicts the energy-delay tradeoff for several algorithms. *(i)* It is worthy of notice that most of CPU and total energy saving (55% CPU, 50% total energy saving) can be obtained by trading only 10 minutes delay. This is due to the fact that CPU power consumption is modeled by a cubic polynomial scaling of CPU speed such as (26). Therefore, by smoothing CPU speed along with time slot t, CPU power can be saved. However, since the smoothing CPU speed makes the system be insensitive to queue variation, average delay would be longer. Network power consumption can be saved until 40% with 20 minutes transmission delay. This power saving comes from the fact that the smartphone is reluctant to transmit data through 3G which is energy-inefficient network than WiFi, yet the smartphone would wait for WiFi network. *(ii)* SpeedControl algorithm well catches up with the delay performance and energy consumption of an optimal algorithm (OPT) even though SpeedControl is much simpler algorithm than the optimal algorithm. *(iii)* SpeedControl saves 56%, 52% (in CPU), 30%, 33% (in total) energy for 10 minutes average delay than Max+SALSA and DVFS+SALSA, respectively. The power saving gains of SpeedControl come from the fact that the

[3]We also use this estimation method in experiment

[4]In this algorithm, CPU speed is maximum when CPU workload is greater than a threshold, and linearly decrease when CPU workload is less than the threshold which can be manually controllable.

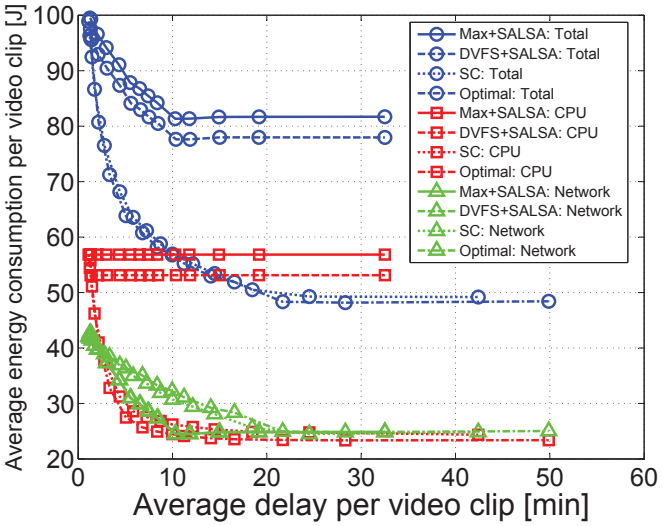


Fig. 4: Energy-delay tradeoff of different algorithms: WiFi temporal coverage - 65%, processing density (NA,NNA) = (300cycles/bit,1200cycles/bit)

algorithm pushes NA workloads from CPU side to network side only when the network side requires the workloads. This implies that joint consideration of application scheduling, CPU speed and network selection is imperative for optimizing CPU and network power in smartphone.

Impact of processing density, arrival rate and WiFi temporal coverages. Fig. 5 presents total (CPU+network) energy consumption of the combination of existing algorithms (DVFS+SALSA, Max+SALSA), normalized by total energy consumption of SpeedControl as a function of average delay of NA and NNA per video clip. For this simulation, we generate WiFi temporal coverage trace for different WiFi coverages using measured WiFi temporal coverage distribution and uplink throughput. *(i)* As processing density of NA is smaller, and arrival rate of NA is higher, SpeedControl obtains more energy saving gain. The gain from low processing density of NA is because that SpeedControl quickly responses to the needs of network side. Also, the gain from high arrival rate comes from the fact that CPU side exactly know when network side needs the workload to transmit in SpeedControl whereas the CPU side does not know what happens at the network side in DVFS+SALSA and Max+SALSA. *(ii)* As arrival rate of NNA is smaller, SpeedControl obtains more energy saving gain. This implies that NA gives more energy saving impact on SpeedControl than NNA. *(iii)* As WiFi temporal coverage is wider, SpeedControl achieves more energy saving. This is due to the fact that wider WiFi coverage makes the smartphone users exploit more energy-efficient network, i.e., WiFi network, thus, the average networking power is reduced, which means that CPU power consumption is bigger impact on total energy consumption than network power. Since our SpeedControl algorithm obtains higher energy saving in CPU part than network part as shown in Fig. 4, total energy saving of SpeedControl increases compared to other algorithms as the WiFi temporal coverage increases.

C. Experiment

Setup. We develop a prototype of SpeedControl application which adopts our SpeedControl algorithm using Android software development kit (SDK) based on the open source code of NSTools application [21] which enables to control CPU clock manually. For estimation of uplink throughput, our private server transmits acknowledgement (ACK), which contains uplink throughput information, to device every 5 seconds. For experiment, we prepare rooted smartphone (Nexus S) and Monsoon power monitor [19]. A smartphone runs two applications: *(i)* video encoding application, *(ii)* prototype of our SpeedControl application which selects CPU speed and network interface and then transmit the encoded data to our server, yet they can be seen as one networking application. We consider that 5 video clips (21MByte per one clip) are arrived at the specific time. Also, the smartphone is associated with one WiFi AP[5], and the smartphone is connected to Monsoon power monitor to measure the total energy. As performance metrics, we measure *(i)* battery level by application (visualized as % or bar in most of smartphones) and real power using Monsoon power monitor and *(ii)* the average delay of video clips when 4 video clips are fully transmitted.

Observation. We obtain three observations from experiment in Fig. 6. *(i)* (Fig. 6(a)) DVFS+SALSA and Max+SALSA consume about 70% and 80% more energy than SpeedControl algorithm with the same delay (about 11 minutes) in real power measurement for transmitting 4 video clips, respectively. *(ii)* (Fig. 6(b)) Our SpeedControl consumes 50% and 40% less battery than Max+SALSA and DVFS+SALSA by trading similar delay, respectively. *(iii)* Smartphone users who install our SpeedControl application save 10% of battery level (spend 10% of battery) for uploading 4 video clips (total 84MBytes) by permitting about 3 minutes more delay when the starting battery level is 70%.

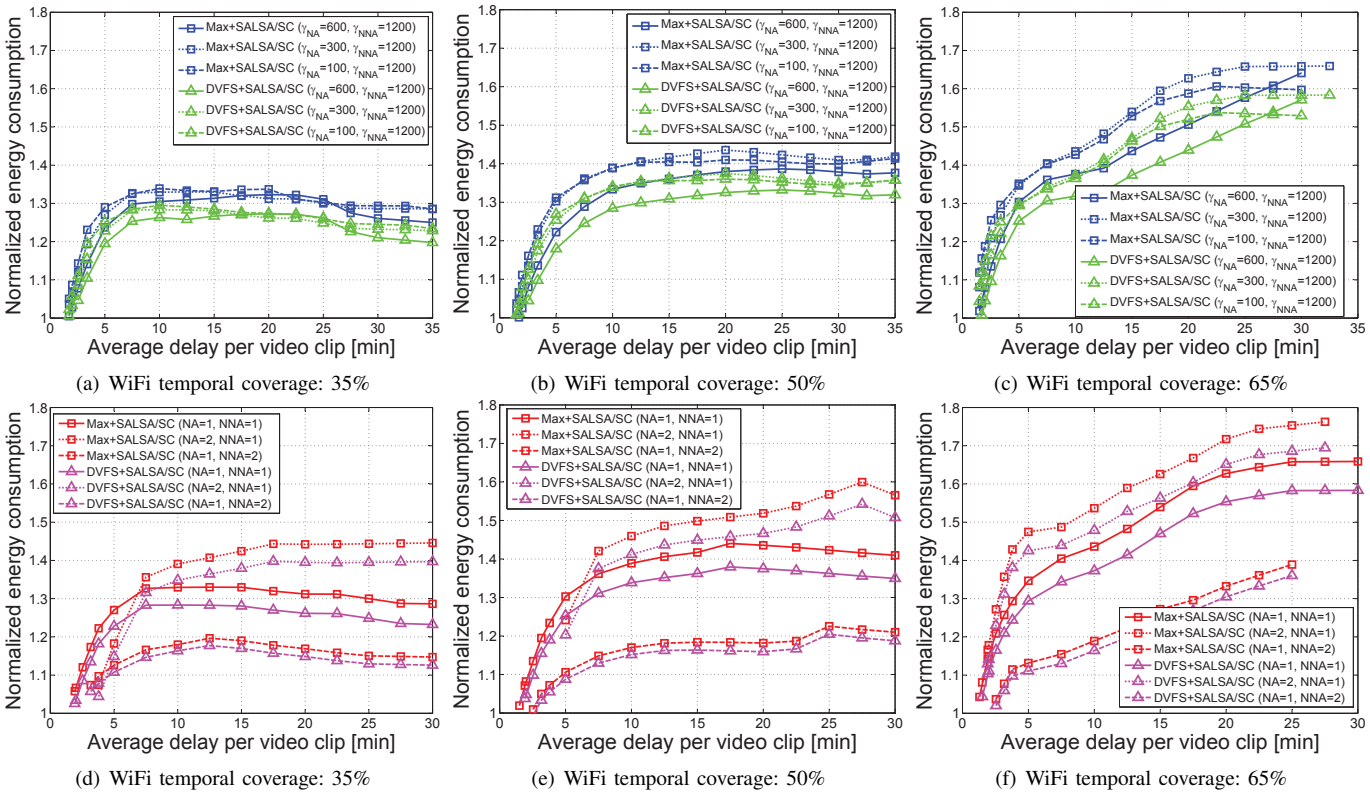
V. Conclusion

In this paper, we investigate key processing and networking features of contemporary smartphones in terms of tradeoff between energy and delay. Based on this study, we suggest SpeedControl algorithm, which jointly optimizes CPU speed and network (wireless interface) selection so as to answer how much energy can be saved further by the joint optimization. SpeedControl well isolates the performance of non-networking applications from that of networking application as well as obtains high energy saving by trading small delay. Finally, through extensive simulations and experiment studies including meaningful real measurement results, we made several important observations which provide us with a message that joint optimization of CPU and network speed would be imperative, especially in future network trend where the more energy-efficient networks are deployed.

VI. Acknowledgements

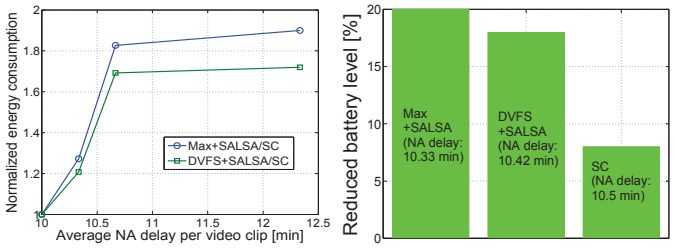
The authors would like to thank Hong-hwan Jeon in heaven for help with 3G/WiFi throughput measurement study.

[5]This WiFi AP is private AP for only one experimental smartphone.



(a) WiFi temporal coverage: 35%
(b) WiFi temporal coverage: 50%
(c) WiFi temporal coverage: 65%
(d) WiFi temporal coverage: 35%
(e) WiFi temporal coverage: 50%
(f) WiFi temporal coverage: 65%

Fig. 5: Normalized energy consumptions for several WiFi temporal coverages: (a)-(c) are the results under the same arrival rate (NA:NNA=1:1), (d)-(e) are the results under the same processing density (γ_{NA} : γ_{NNA}=300cycles/bit:1200cycles/bit)



(a) Energy-delay tradeoff
(b) Reduced battery level

Fig. 6: Experimental results for several algorithms

REFERENCES

[1] "Snapdragon S4 processors: System on chip solutions for a new mobile age, White Paper," pp. 1–8, 2011.

[2] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Proc. of USENIX*, 2012.

[3] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. of USENIX*, Boston, MA, USA, Jun. 2010, pp. 21–21.

[4] T. K. J. Chen, C. Yang and C. Shih, "Energy-efficient real-time task scheduling in multiprocessor DVS systems," in *Proc. of Asia and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2007, pp. 342–349.

[5] Y. Liang, P. Lai, and C. Chiou, "An energy conservation DVFS algorithm for the android operating system," *Journal of Convergence*, vol. 1, no. 1, pp. 93–100, Dec. 2010.

[6] M. Ra, J. Peak, A. Sharma, R. Govindan, M. Krieger, and M. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proc. of MobiSys*, SF, California, USA, Jun. 2010, pp. 255–270.

[7] 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 *Proc. of IEEE INFOCOM*, Turin, Italy, Apr. 2013, pp. 14–19.

[8] "Google Play." [Online]. Available: https://play.google.com/store/apps/collection/topselling_free

[9] M. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, pp. 1–211, 2010.

[10] K. Son and B. Krishnamachari, "Speedbalance: speed-scaling-aware optimal load balancing for green cellular networks," in *Proc. of IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2816–2820.

[11] A. Wierman, L. L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 2007–2015.

[12] M. Andrews, A. Anta, L. Zhang, and W. Zhao, "Routing for energy minimization in the speed scaling model," in *Proc. of IEEE INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 1–9.

[13] "Kernel governors, modules, I/O scheduler, CPU tweaks AIO app configs." [Online]. Available: http://forum.xda-developers.com/showthread.php?t=1369817

[14] A. Rahmati and L. Zhong, "Context-for-wireless: context-sensitive energy-efficient wireless data transfer," in *Proc. of MobiSys*, San Juan, Puerto Rico, Jun. 2007, pp. 165–178.

[15] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: how much can wifi deliver?" *IEEE/ACM Trans. Networking*, vol. 21, no. 2, pp. 536–550, Apr. 2013.

[16] M. Shin, S. Chong, and I. Rhee, "Dual-resource TCP/AQM for processing-constrained networks," *IEEE/ACM Trans. Networking*, vol. 16, no. 2, pp. 435–449, Apr. 2008.

[17] J. Kwak, O. Choi, S. Chong, and P. Mohapatra, "Dynamic speed scaling for energy minimization in delay-tolerant smartphone applications," *Technical Report*, Jul. 2013. [Online]. Available: http://netsys.kaist.ac.kr/publication/papers/Resources/R6

[18] L. Georgiadis, M. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Networking*, vol. 1, no. 1, pp. 1–149, 2006.

[19] "Power Meter Device: Monsoon Power Monitor." [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/

[20] "Dataset for statistics and social network of YouTube videos." [Online]. Available: http://netsg.cs.sfu.ca/youtubedata/

[21] "NSTools application and open source code v1.16." [Online]. Available: http://forum.xda-developers.com/showthread.php?t=1333696